大数据分析五个重要阶段

- 数据获取和记录
- 数据抽取、清洗和注记
- 数据集成、聚集和表示
- 数据分析和建模
- 数据解释

大数据分析四个应用实例

- XML文档存储与检索系统
- 股票市场预测系统
- 海量视频检索系统
- HDFS云文件系统

大数据分析技术

- Hadoop
- HDFS
- HBase
- MapReduce

实战大数据

怎么做大数据分析？大数据分析怎么应用在业务系统上？

鲍亮 李倩 编著



清华大学出版社

实战大数据

鲍亮 李倩 编著



清华大学出版社
北 京

内 容 简 介

“数据是重要资产”已成为大家的共识，众多公司都在争相分析、挖掘大数据背后的信息资源。本书在此背景下，对目前大数据及其相关技术的发展进行总结，理论联系实践，既不缺乏理论深度又具有实用价值。

本书共 12 章，内容包括大数据的概念、特点、发展历史，数据获取与存储，数据抽取和清洗，数据集成，数据的查询、分析与建模，异构数据采集，文档的存储与检索，异种数据的统一访问与转换，基于微博的股票市场预测系统实例，海量视频检索系统实例，HDFS 云文件系统实例。

本书适合大数据技术初学者、大数据从业人员和研究人员，也可以作为高等院校相关专业师生的教学参考书。

图书在版编目（CIP）数据

实战大数据 / 鲍亮，李倩编著. – 北京：清华大学出版社，2014

ISBN 978-7-302-34866-5

I. ①实… II. ①鲍… ②李… III. ①数据处理－研究 IV. ①TP274

中国版本图书馆 CIP 数据核字（2013）第 310963 号

责任编辑：夏非彼
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社

网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**33.75　　**字　　数：**864 千字
版　　次：2014 年 3 月第 1 版　　**印　　次：**2014 年 3 月第 1 次印刷
印　　数：1~3500
定　　价：79.00 元

产品编号：000000-01

前　言

大数据时代已经到来，大数据处理已经成为当今信息处理的热点研究内容。不同于大规模数据，大数据具有自身鲜明的 4V 特征：Volume（规模性）、Variety（多样性）、Velocity（高速性）和 Veracity（真实性）。大数据不仅规模大，更需要采取新的数据思维来应对，其必然导致理论和技术上的革新。因此，大数据分析也被认为是继实验、理论和计算之后的科学研究第四范式。大数据的出现必将颠覆传统的数据管理方式，在数据来源、数据处理方式和数据思维方面都会对其带来革命性的变化。

2013 年初，美国计算机协会数据库专家委员会联合研究界、产业界和政府部门的相关研究人员，发布了大数据研究白皮书，提出了大数据分析的 5 个重要阶段：数据获取和记录，数据抽取、清洗和注记，数据集成、聚集和表示，数据分析和建模，数据解释。在这 5 个阶段中需要考虑数据的异构性、规模、时效性、复杂性和隐私问题。本书以此为提纲进行内容组织，首先介绍了 5 个阶段中相关的科学与技术问题，然后以实际案例的形式详细介绍了数据采集、数据存储与检索、数据处理、数据访问与转换 4 个大数据领域的重要问题，最后以股票市场预测系统、海量视频检索系统和云文件系统 3 个大数据实际应用系统为例详细介绍如何进行问题分析、数据建模以及系统的设计与实现。本书强调理论联系实际，重点在于介绍如何利用现有技术解决实际的大数据问题。

目前市场上以大数据为主题的书籍较多，但经过作者调研，未见以“利用现有技术解决大数据问题”为主题的大数据实战类书籍。本书编写团队核心成员自 2010 年起陆续承担了一些与大数据采集、存储、处理、分析、挖掘和检索方面的研究与应用开发工作，具有丰富的项目实践经验。这些实际项目经验形成了本书最为核心的第 6~12 章的内容。通过项目实战，我们积累了一些解决大数据问题的宝贵经验，对大数据的核心技术有了较为深刻的理解，认为有必要将自己的经验和认识整理出来，以满足广大读者利用现有技术解决大数据实际问题的迫切需求与心情，这也是书名的由来。

本书适合不同层次的读者阅读，建议读者根据自己的兴趣和目的有选择性地阅读：希望了解大数据相关的基础理论与技术的读者，可以重点阅读第 1~5 章；对于大数据领域的初学者，可以重点阅读第 1~9 章；对于已经掌握大数据基础理论，具有一定的技术基础，想解决实际

大数据问题的读者，可以重点阅读第 10~12 章。

除封面署名的作者之外，这里还需要感谢书籍编写团队核心成员李江、张翔、杨阳、王贺、刘凯、王学良、张静、周文琳、刘晓静、张艳华、王炎楠、黄鹏、高小青的辛勤努力。还需要感谢阚传奇、蒋帆的大力帮助，感谢我的导师陈平教授在大数据科学研究方面对我的启发与悉心指导。

由于大数据涉及的学科面很广，研究问题纷繁复杂，相关资料目前还比较少，加之作者水平有限，时间紧迫，书中难免存在错误与不当，恳请读者批评指正。建议和意见请发至作者邮箱 baoliang@mail.xidian.edu.cn。

编者

2013 年 12 月

目　录

第一篇　大数据基础篇

第二篇 大数据深入篇

第一篇

大数据基础篇



第 1 章

大数据介绍

IT 行业总不乏新鲜的主题，而大数据正当其兴，被业界热情传诵。"数据是重要资产"这一概念已成为大家的共识，众多公司争相分析、挖掘大数据背后的重要资源。为了帮助读者理解大数据的来龙去脉，本章将从大数据的历史与发展、大数据的定义、大数据的研究内容、大数据问题在国内外政府、公司和大学的研究现状等方面进行论述，为这一新兴概念勾勒出一个雏形。

1.1 大数据相关概念

1.1.1 大数据的历史

大数据（Big Data）目前已经成为 IT 领域最为流行的词汇，其实它并不是一个全新的概念。早在 1980 年，著名未来学家阿尔文・托夫勒便在《第三次浪潮》一书中，明确提出"数据就是财富"这一观点，并将大数据热情地赞颂为"第三次浪潮的华彩乐章"。

大数据中的"大"是一个相对概念，数据库、数据仓库、数据集市等信息管理领域的技术，很大程度上也是为了解决大规模数据的问题。被誉为数据仓库之父的 Bill Inmon 早在 20 世纪 90 年代就经常将"大数据"这一概念挂在嘴边了。

目前得到广泛认可的大数据概念首先由知名咨询公司 Gartner 的一位资深分析师 Douglas Laney 提出。他于 2001 年在 *Application Delivery Strategies* 上撰写了一篇名为"3D Data Management: Controlling Data Volume, Velocity, and Variety"的文章，指出大数据管理面临三个 V 的挑战：数据量（Volume）、数据多样性（Variety）、高速（Velocity）。"3V"后来成为大数据公认的三个基本特征。随后，Gartner 发布了大数据的模型，强调大数据需要管理采用传统数据管理技术无法管理的数据，比如微博数据、海量交易数据、多媒体数据，等等。

2008 年 9 月，自然杂志推出《大数据》专刊，通过"The next Google"、"Data wrangling"、"Welcome to the petacentre"、"Distilling meaning from data"等多篇文章，全方位介绍了大数据问题的产生及对各个研究领域的影响，首次将"大数据"这一概念引入科学家和研究人员的视野。

2009 年 8 月，Adam Jacobs 在 *ACM Queue* 上发表文章"The Pathologies of Big Data"，文章讨论了大数据问题的起源、发展与现状，指出"大数据"这一概念是相对的，并提出应该考虑为什

么会出现“大数据”这一现象、“大数据”产生的很大一部分原因是数据录入更加容易等观点。

2011 年 2 月 11 日的《科学》杂志专门推出《数据处理》（*Dealing with Data*）专刊，对大数据现象在科学领域的现状进行了全面分析。该专刊首先联合《科学》杂志的兄弟期刊 *Science Signaling*、*Science Translational Medicine* 和 *Science Careers*，展开了对各科学领域研究数据规模急剧增大情况下各种问题的调研，问题包括“研究数据的规模”、“研究数据如何存储”，等等。随后，该专刊发表多篇文章，对天文学、气象学、生态学、神经科学、信号处理、社会科学、生物学等多个学科的大数据问题进行了解释和阐述，内容涵盖数据采集、分析、处理、挖掘和可视化等多个方面。

2011 年 5 月，麦肯锡全球研究院发表 *Big data: The next frontier for innovation, competition, and productivity* 白皮书，指出企业正在面临海量的交易数据、顾客信息、供货商信息和运营数据等，需要对这些数据进行管理与挖掘。在物联网环境下，传感器、智能手机、工业设备等都在产生海量数据。互联网中的多媒体数据量也在以指数级上升，如何处理这些数据，为用户提供有用的信息，成为需要考虑的重要问题。

2011 年 5 月 26 日，经济学人发表“Building with big data”指出在数据极度膨胀的时代，要掌握数据的分析与处理能力，成为数据的主人，而不要成为数据的奴隶。

2012 年 2 月 11 日，纽约时报发表“The Age of Big Data”，向大众宣传大数据时代的到来。

2012 年 3 月 22 日，奥巴马宣布以 2 亿美元投资大数据领域，在次日的电话会议上，美国政府将数据定义为“未来的新石油”，美国政府认识到了一个国家拥有数据的规模、活性及解释运用的能力将成为综合国力的重要组成部分，未来对数据的占有和控制甚至将成为继陆权、海权、空权之外的另一种国家核心资产。

2012 年 7 月 10 日，联合国在纽约总部发布了一份大数据政务白皮书，总结了各国政府如何利用大数据更好地服务和保护人民。

1.1.2　大数据的定义

1. 维基百科的定义

大数据是指其大小或复杂性无法通过现有常用的软件工具，以合理的成本并在可接受的时限内对其进行捕获、管理和处理的数据集。这些困难包括数据的收入、存储、搜索、共享、分析和可视化。

2. Garnter 的定义

Garnter 咨询公司关注大数据的三个量化指标：数据量、数据种类和处理速度。一般企业所面对的数据管理管理的是数据库、结构化数据，以及所能预先安装好的管理软件所带来的数据。大数据管理的往往是我们无法管理的数据，比如来自企业外部，微博、社交网站和多媒体等各种载体的数据。

数据多样性将是大数据的一个重点。它意味着未来数据的产生将更加方便、快捷，无所不在。

数据种类随着物联网等技术的不断兴起而飞速增加，特别是以多媒体数据为代表的非结构化数据迅速增加，为大数据的分析与处理带来了很大难度。处理速度与企业 CIO 关注的系统性能不是等同的关系。这里的速度指的是从数据产生到最终针对数据产生决策的速度，包括存储的过程、计算的过程、系统模型和以什么方式提交出最后的结果。因此，速度不仅是计算能力和存储性能的问题，还要考虑数据管理、数据保护等方面的响应与处理速度。在大数据问题中，速度往往是性命攸关的。比如对于灾难的预测，当灾难发生时，要很快对灾难发生的程度、影响的区域范围、对长远的影响等量化出来。这是大数据很典型的应用，如果短时间内没有计算出来，那么数据就没用了。

另一方面，Garnter 认为在越来越大的数据集上工作能够得到更大的好处，大数据的数据增长挑战（或机遇）是三维立体的：不断增长的数据量、不断增加的速率（数据 I/O 的速度）和不断增加的种类（数据类型、数据源）。而传统的存储技术难以应对大数据处理的三大挑战。

- 挑战一：不断增长的数据量。在大数据背景下，数据通常是不能删除的，这是企业的宝贵的财富，因此数据将不断积累增长。与此同时，增长有加速的趋势，经常会超出人们预计或规划，从而对信息系统带来了极大的挑战。信息中心需要管理 TB 级甚至 PB 级数据。要为这些数据提供存储、保护和使用的方案，IT 系统需要不断地做相应升级或重构，需要投入大量人力物力。
- 挑战二：多格式数据。海量数据包括了越来越多不同格式的数据，这些不同格式的数据也需要不同的处理方法。从简单的电子邮件、数据日志和信用卡记录，再到仪器收集到的科学研究数据、医疗数据、财务数据以及丰富的媒体数据（包括照片、音乐、视频等），都具有这个特点。比如视频文件格式就非常多，有各软件厂商的厂商标准的格式，工业标准组织的工业标准格式。各种格式在当前高清化的趋势下，数据粒度更小，处理更精细，更复杂的格式还不断出现，造成单一文件的体积成倍增加，从而要求处理速度也成倍增加。
- 挑战三：性能。速度是指数据从客户端到处理器和存储的移动速度，涉及终端数据处理能力、数据流访问和交付、服务器计算处理能力以及后端存储的吞吐能力。速度意味着要求数据必须以多快的频率被处理。大数据处理需要不同于交易类应用的速度，通常其对带宽的要求比 IO 操作的速度更重要。

3. IBM 对大数据的定义

IBM 专门开辟了大数据专栏，从大数据的定义、大数据处理平台等多个方面对大数据问题及解决方案进行了阐述。

Big data spans three dimensions: Volume, Velocity and Variety.

Volume: Enterprises are awash with ever-growing data of all types, easily amassing terabytes—even petabytes—of information.

Velocity: Sometimes 2 minutes is too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

Variety: Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more. New insights are found when analyzing these data types together.

IBM 认为大数据横跨三个层面：规模、速度和种类。

- 规模：企业充斥着日益增长的所有数据类型，容易积累 TB 级甚至 PB 级的信息数据。
- 速度：对于有些应用来说，两分钟的处理时间都为时已晚。对于欺诈追踪等时间敏感的处理流程而言，大数据必须快速流入企业信息系统，并得到快速处理，以最大化其价值。
- 种类：大数据可以是任何类型的数据，包括文本、传感器数据、音频、视频、单击流、日志文件等结构化和非结构化数据。当能够一起分析这些类型的数据时，就可以得到新的见解。

从上述定义可以看出，IBM 把大数据概括为三个 V，即大规模（Volume）、高速度（Velocity）和多样化（Variety），这些特点也反映了大数据所潜藏的价值（Value，第四个“V”）。因此大数据的特征可以整体概括为：“海量+多样化+快速处理+价值”。

4. 微软对大数据的定义

微软在 SQL Server 产品网站上开辟专栏，给出了大数据的相关概念，强调需要将大数据转化为企业的洞察力。

Big data is the increasingly large and complex data that is now challenging traditional database systems

Data **volume** is exploding: In the last few decades computing and storage capacity have grown exponentially, driving down cost to near zero. The rise of new technologies like Hadoop is significantly changing the economics of large scale data processing by enabling customers to analyze petabytes of data with industry standard hardware. According to IDC the digital universe will grow to 35 zettabytes (i.e. 35 trillion terabytes) globally by 2020.

The **variety** of data is increasing. It's all getting stored and nearly 85 percent of new data is unstructured data. The real questions now are: How do you put all this captured and stored data to good use? How do you analyze it to make better decisions?

The **velocity** of data is speeding up the pace of business. Data capture has become nearly instantaneous thanks to new customer interaction points and technologies. Real-time analytics is more important than ever.

微软对大数据的定义也采用了“3V”模型，并进一步指出：（1）由于硬件成本的持续降低和新型数据源（RFID、互联网和社交媒体等）的加入，数据量会持续增加；（2）文本、博客、视频、图片、购买历史等多样化的数据大大增加了数据的种类，数据具有鲜明的多样性特征；（3）随着网站、ATM 取款机、POS 收款机等设备成为大数据的数据源，数据产生速度飞速增加。

5. SAS 对大数据的定义

作为专业的商业分析软件与服务供应商，SAS 在大数据传统“3V”模型定义的基础上加入了

“可变性”和“复杂性”两个重要特征。

- **Variability.** In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something big trending in the social media? Perhaps there is a high-profile IPO looming. Maybe swimming with pigs in the Bahamas is suddenly the must-do vacation activity. Daily, seasonal and event-triggered peak data loads can be challenging to manage – especially with social media involved.

- **Complexity.** When you deal with huge volumes of data, it comes from multiple sources. It is quite an undertaking to link, match, cleanse and transform data across systems. However, it is necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control. Data governance can help you determine how disparate data relates to common definitions and how to systematically integrate structured and unstructured data assets to produce high-quality information that is useful, appropriate and up-to-date

可变性主要反映了数据流可能具有高度的不一致性，并存在周期性的峰值。例如社交网络中的某个热点趋势可能是一次高收益的 IPO。对日常的、季节性和时间驱动的峰值数据流的管理具有挑战性，特别是当社交媒体介入的情况下。

复杂性主要体现在数据来源的多样性上。连接、匹配、清洗和转化来自多个系统的数据是一件非常复杂的事情。除此以外，还需要考虑不同数据源之间的连接关系、关联关系和层次关系等。需要实施数据治理策略，帮助企业系统地集成结构化和非结构化数据资产，产生高质量、恰当的、最新的有用信息。

1.2 大数据研究内容

2012 年冬季，来自 IBM、微软、谷歌、HP、MIT、斯坦福、加州大学伯克利大学、UIUC 等产业界和学术界的数据库领域专家通过在线的方式共同发布了一个关于大数据的白皮书：“Challenges and Opportunities with Big Data”。该白皮书首先指出大数据面临着 5 个主要问题，分别是异构性（Heterogeneity）、规模（Scale）、时间性（Timeliness）、复杂性（Complexity）和隐私性（Privacy）。在这一背景下，大数据的研究工作将面临 5 个方面的挑战：

- 数据获取问题。数据海啸需要我们对“哪些数据需要保存，哪些数据需要丢弃，如何可靠地存储我们需要的数据（同时存储该数据正确的元数据）”等问题进行决策，目前这些决策还只能采用特定方法（ad hoc）给出。
- 数据结构问题。tweet 和 blog 是没有结构的数据；图像和视频在存储和显示方面具有结构，但无法包含语义信息并进行检索。如何将这种没有语义的内容转换为结构化的格式，并进

行后续处理，是需要应对的另外一项重要挑战。

- 数据集成问题。只有将数据之间进行关联，才能充分发挥数据的作用，因此数据集成也是一项挑战。
- 数据分析、组织、抽取和建模是大数据本质的功能性挑战。数据分析是许多大数据应用的瓶颈，目前底层算法缺乏伸缩性、对待分析数据的复杂性估计不够。
- 如何呈现数据分析的结果，并与非技术领域的专家进行交互。

白皮书对大数据的“3V”模型进行了解释，指出现有的工作对数据的隐私性和易用性方面考虑不周。另外，大数据的分析包含多个步骤，目前的研究大多关注数据建模和分析，而对其他阶段关注不够。即使在数据分析阶段，目前的研究仍然没有很好地理解数据建模与分析在多租户集群环境中的复杂性，在该环境中，多个用户程序会并发执行。

为了应对上述挑战，白皮书建议采用现有成熟技术解决大数据带来的挑战，并给出了大数据分析的分析步骤，如图 1.1 所示。

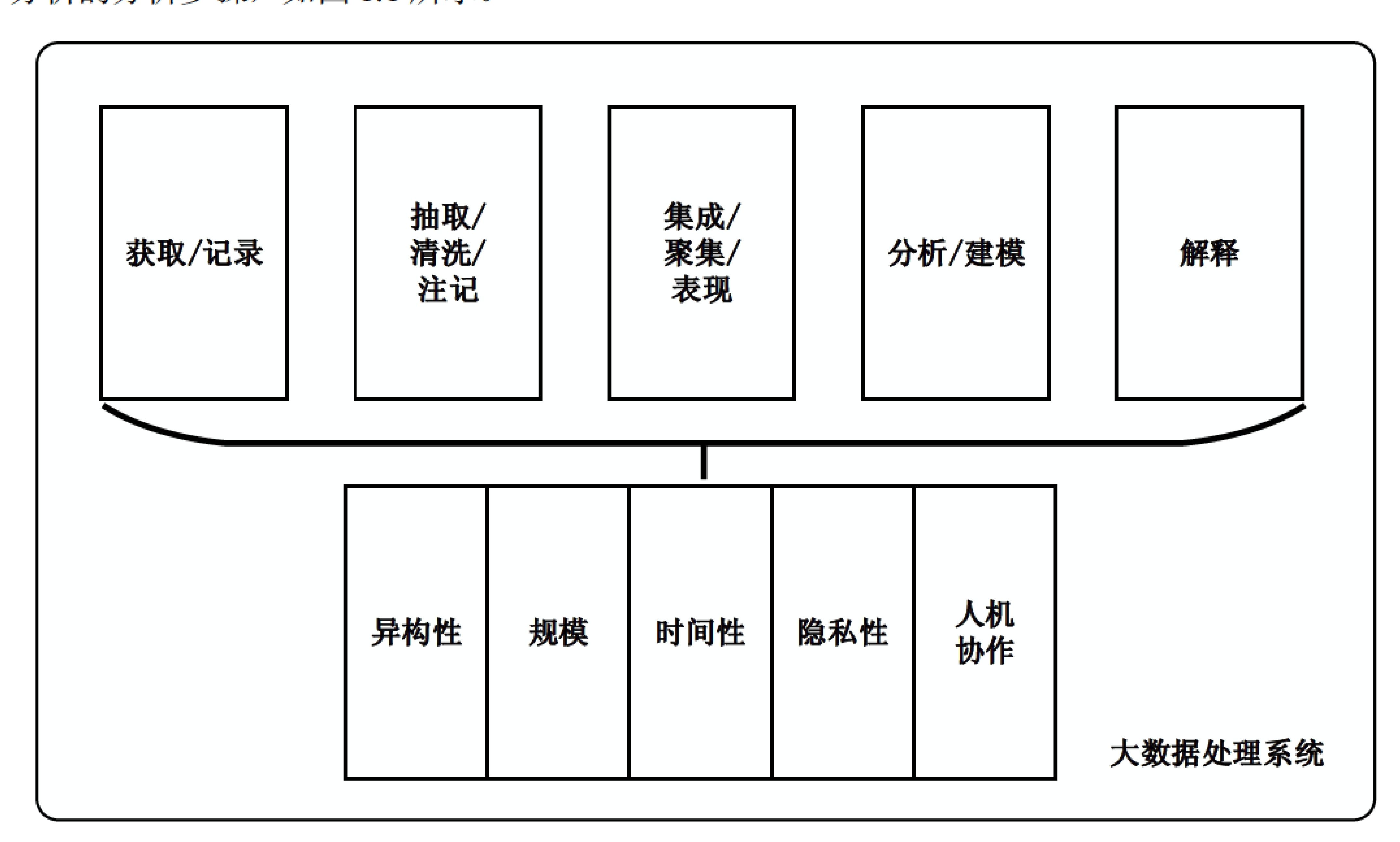


图 1.1　大数据处理参考框架

从图 1.1 中可以看出，大数据处理过程可以大致分为数据获取/记录、信息抽取/清洗/注记、数据集成/聚集/表现、数据分析/建模和数据解释 5 个主要阶段，贯穿所有节点，系统需要考虑数据的异构性、规模、时间性、隐私性和人机协作等方面的因素。在每一个阶段，都面临着各自的研究问题与挑战。

1. 数据获取和记录（Data Acquisition and Recording）

面临的挑战包括：

- 如何对原始数据进行智能化处理，过滤不需要的数据；

- 在线处理技术，直接对数据进行处理，而不需要存储后再进行过滤；
- 自动生成正确的元数据，描述记录了什么数据以及数据的记录和度量方式。

可能的研究方向：

- 研究数据压缩（reduction）中的科学问题，能够智能地处理原始数据，在不丢失信息的情况下，将海量数据压缩到人可以理解的程度；
- 研究“在线”数据分析技术，能够处理实时流数据；
- 研究元数据自动获取技术和相关系统；研究数据来源（data provenance）技术，追踪数据的产生和处理过程。

2. 信息抽取和清洗（Information Extraction and Cleaning）

一般来说，收集到的信息通常不能直接用来进行数据分析，而需要一个信息抽取过程，将需要的信息从底层数据源中抽取出来，形成适于分析的结构，完成这样的工作需要持续的技术挑战。抽取的对象可能包含图像、视频等具有复杂结构的数据，而且该过程通常是与应用高度相关的。除此以外，由于监控摄像头、装载有 GPS 的智能手机、相机和其他便携设备无处不在，丰富的、高保真度的位置和轨迹数据也应该被收集与处理。

一般认为，大数据通常会反映事实情况，实际上大数据中广泛存在着虚假数据。关于数据清洗的现有工作通常假设数据是有效的、组织良好的，或对其错误模型具有良好的先验知识，这些假设在大数据领域将不再正确。

3. 数据集成、聚集和表现（Data Integration, Aggregation, and Representation）

由于大量异构数据的存在，大数据处理不能只对数据进行记录，然后就将其放入存储中。如果仅仅是将一堆数据放入存储中，那么其他人就可能无法查找改数据，更不用说使用数据了。即使各个数据源都存在元数据，将异构数据整合在一起仍然是一项巨大的挑战。

对大规模数据进行有效分析需要以自动化的方式对数据进行定位、识别、理解和引用。为了实现该目标，需要研究数据结构和语义的统一描述方式与智能理解技术，实现机器自动处理，从这一角度看，对数据结构与数据库的设计也显得尤为重要。

4. 查询处理、数据建模和分析（Query Processing, Data Modeling, and Analysis）

查询和挖掘大数据的方法，从根本上不同于传统的、基于小样本的统计分析方法。大数据中的噪声数据很多，具有动态性、异构性、相互关联性、不可信性等多种特征。尽管如此，即使是充满噪声的大数据也可能比小样本数据更有价值，因为通过频繁模式和相关性分析得到的一般统计数据通常强于具有波动性的个体数据，往往透露更可靠的隐藏模式和知识。此外，互联的大数据可形成大型异构的信息网络，可以披露固有的社区，发现隐藏的关系和模式。此外，信息网络可以通过信息冗余以弥补缺失的数据、交叉验证冲突的情况、验证可信赖的关系。

数据挖掘需要完整的、经过清洗的、可信的、可被高效访问的数据，以及声明性的查询（例如 SQL）和挖掘接口，还需要可扩展的挖掘算法及大数据计算环境。与此同时，数据挖掘本身也

可以提高数据的质量和可信度，了解数据的语义，并提供智能查询功能。

大数据也使下一代的交互式数据分析实现实时解答。未来，对大数据的查询将自动生成网站上创作的内容、形成专家建议，等等。在 TB 级别上的可伸缩复杂交互查询技术是目前数据处理的一个重要的开放性研究问题。

当前大数据分析的一个问题是缺乏数据库系统之间的协作，这些数据库存储着数据并提供 SQL 查询，而且具有对多种非 SQL 处理过程（例如数据挖掘、统计等）支持的工具包。今天的数据分析师一直受到“从数据库导出数据，进行数据挖掘与统计（非 SQL 处理过程），然后再写回数据库”这一烦琐过程的困扰。现有的数据处理方式是前述的交互式复杂处理过程的一个障碍，需要研究并实现将声明性查询语言与数据挖掘、数据统计包有机整合在一起的数据分析系统。

5. 解释（Interpretation）

仅仅有能力分析大数据本身，而无法让用户理解分析结果，这样的效果价值不大。如果用户无法理解分析，最终，一个决策者需要对数据分析结果进行解释。对数据的解释不能凭空出现，通常包括检查所有提出的假设并对分析过程进行追踪。此外，分析过程中可能引入许多可能的误差来源：计算机系统可能有缺陷、模型总有其适用范围和假设、分析结果可能基于错误的数据，等等。在这种情况下，大数据分析系统应该支持用户了解、验证、分析计算机所产生的结果。大数据由于其复杂性，这一过程特别具有挑战性，是一项重要的研究内容。

在大数据分析的情景下，仅仅向用户提供结果是不够的。相反，系统应该支持用户不断提供附加资料，解释这种结果是如何产生的。这种附加资料（结果）称之为数据的出处（data provenance）。通过研究如何最好地捕获、存储和查询数据出处，同时配合相关技术捕获足够的元数据，就可以创建一个基础设施，为用户提供解释分析结果，重复分析不同假设、参数和数据集的能力。

具有丰富可视化能力的系统是为用户展示查询结果，进而帮助用户理解特定领域问题的重要手段。早期的商业智能系统主要基于表格形式展示数据，大数据时代下的数据分析师需要采用强大的可视化技术对结果进行包装和展示，辅助用户理解系统，并支持用户进行协作。

此外，通过简单的单击操作，用户应该能够向下钻取到每一块数据，看到和了解数据的出处，这是理解数据的一个关键功能。也就是说，用户不仅需要看到结果，而且需要了解为什么会产生这样的结果。然而，数据的原始出处（特别是考虑到整个分析过程具有管线结构）对于用户来说技术性太强，无法抓住数据背后的思想。基于上述问题，需要研究新的交互方式，支持用户采用“玩”的方式对数据分析过程进行小的调整（例如对某些参数进行调整，等等），并立即对增量化的结果进行查看。通过这种方法，用户能够对分析结果有一个直观的理解，从而更好地理解大数据背后的价值。

1.3 大数据研究现状

1.3.1 学术界现状

1. 国外学术界大数据研究现状

（1）MIT

2012 年 5 月 31 日，MIT 计算机科学和人工智能实验室（CSAIL）与英特尔联合成立了“bigdata@CSAIL”大数据研究项目。该项目主要关注大数据在计算平台、可伸缩的算法、机器学习和理解、隐私和安全 4 个方面的科学问题与解决方案。该项目汇聚了 CSAIL 中以 Sam Madden 为代表的 29 位研究者，分别从系统风险分析、智能城市、数据存储、机器学习算法、信用记录分析、交互式数据可视化、计算机系统结构仿真、下一代搜索引擎等多个子项目入手，从多个方面对大数据问题进行了深入的研究。

（2）加州大学伯克利分校

美国政府于 2012 年 3 月为加州大学伯克利分校注资 1000 万美元，开展 Big Data Research and Development Initiative（大数据研究与开发）项目的研究。该项目旨在采用机器学习技术和云计算技术解决大数据问题，挖掘大数据中的重要信息。

加州大学伯克利分校 Lawrence 国家实验室的研究人员领导着“Scalable Data Management, Analysis, and Visualization”研究中心，该中心联合了 7 所大学和 5 所其他国家实验室，主要从事大数据管理、分析和可视化方面的研究工作。

2012 年 11 月，加州大学伯克利分校开设了一门关于大数据的公开课 Analyzing Big Data With Twitter。该课程由大学教授和 Twitter 技术主管穿插讲解，内容以 Twitter 上面临的实际大数据挑战为蓝本，着重从软件工程的角度介绍大数据的分析技术，探讨解决大数据问题的方法。

2013 年 8 月，加州大学伯克利分校西蒙计算理论研究中心组织了一系列的“大数据研讨会（Big Data Boot Camp）”活动，探索大数据分析与处理过程中的理论计算问题。

（3）斯坦福大学

斯坦福大学医学系专门成立了生物医学专业大数据组，定期组织生物学、医学、计算机等方面的专家就大数据问题进行研讨，旨在跨学科地研究和探讨大数据问题。

在教育培训方面，斯坦福大学提供了大规模数据挖掘（Mining Massive Data Sets）认证课程，学校内的学生可以选修相关课程，获得认证。

（4）华盛顿大学

华盛顿大学计算机科学与工程系利用自身在数据管理、机器学习和开放信息抽取方面的传统优势，开展了研究和学位教育方面的工作。

在研究方面，华盛顿大学计算机科学与工程系展开了大数据管理、数据可视化、大数据系统、

Web 上的大数据、大数据和发现等多项科研项目。

在大数据管理领域，开展了包括 AstroDB、Myria、Nuage、CQMS、Data Eco$y$tem 和 SQLShare6 个有代表性的研究项目，其中 AstroDB 是计算机科学与工程系 2008 年以来一直与华盛顿大学天文学系共同合作的项目，旨在构建能够存储、管理、分析和处理天文学领域大数据的系统。Myria 项目主要关注构建一个快速、灵活的大数据管理系统，将系统以云服务的形式对外暴露。Nuage 项目关注大数据与云计算相关的技术问题，特别关注科学应用问题。CQMS 关注辅助大数据系统使用的相关工具。Eco$y$tem 项目关注大数据市场以及数据管理和定价等方面的问题。SQLShare 是一个基于云计算技术的数据库即服务平台，关注关系数据库自动化使用方面的相关问题，包括安装、配置、数据库模式设计、性能调优和应用构建等问题。

在大数据可视化方面，主要通过设计交互式可视化分析工具，增强数据的分析和交流能力，该项目涉及可视化、交互技术和评估技术的研究与系统实现等方面的问题。

在大数据架构和编程方面，主要研究在计算机系统结构、编程和系统层面上对大数据的支持，主要包括基于 PCM（Phase-Change Memory）的存储系统研究、大规模非规则并行计算（如图分析等）、硬件多线程系统，等等。

在大数据系统方面，主要研究超大规模内存机器、大规模并行系统中的可预测尾延迟（predictable tail-latency）技术等。

在 Web 大数据方面，主要研究 Web 范围内的信息抽取系统，该系统能够读取 Web 上的任意文本数据，抽取有意义的信息，并将其存储到一个统一的知识库中，便于后续的查询工作。

在人才培养和教育方面，计算机科学与工程系于 2013 年 9 月开始招收数据科学的博士学位（特别关注大数据问题）。华盛顿大学将利用整个大学的资源，打造一个跨学科的大数据方面的博士学位。除此以外，华盛顿大学还开设一个关于数据科学方面的认证项目，提供相关的教育与培训服务。

2. 国内学术界大数据研究现状

（1）中国科学院

英特尔公司与中国科学院自动化研究所联合成立“中国英特尔物联技术研究院”，计划未来 5 年投资 2 亿元人民币，着力攻克大数据处理技术、传输技术和智能感知等物联网核心技术。该研究院还将与国际国内一流科研院所、院校和企业合作，建立一个开放式的研究中心。

中国科学院软件研究所 2012 年 5 月 31 日承办了“走进大数据时代研讨会”。国内众多知名大学教授，及行业代表围绕大数据的相关议题展开共同探讨。分析了当前大数据的行业现状，大数据的最新动态及发展趋势。“大数据”概念正在引领中国互联网行业新一轮的技术浪潮。

（2）清华大学

清华大学计算机科学与技术系、地球系统科学研究中心等机构一直从事大数据方向的研究，取得了一些成果，包括清华云存储系统、大数据存储系统、大数据处理平台、社交网络云计算和海量数据处理系统，等等。

2013 年 7 月，人人游戏将向清华大学捐赠 1000 万元，与后者共同建设一个“行为与大数据实验室”。该实验室将主要用于研究网络虚拟社区心理和体验经济心理，为人人游戏的产品开发提

供理论和技术支撑。

（3）北京航空航天大学

在科学研究方面，北京航空航天大学计算学院、爱丁堡大学信息学院、香港科技大学计算机系、宾夕法尼亚大学和百度公司于 2012 年 9 月联合创建“大数据科学与工程”国际研究中心，旨在以当前互联网和大数据时代新型信息技术为牵引，创造新的学术领域和应用增长点。

在人才培养方面，北京航空航天大学计算机学院、北京航空航天大学软件学院、工信部 CSIP 移动云计算教育培训中心于 2013 年联合创办了国内第一个“大数据科学与应用”软件工程硕士专业。该专业以实际需求为牵引，结合企业内训和项目实践，期望学生掌握大数据在数据管理、系统开发、数据分析与数据挖掘等方面的核心技能。

（4）中国人民大学

中国人民大学“云计算与大数据实验室”是由周晓方教授、陆嘉恒副教授领导的，主要关注云计算、非结构化数据、海量数据、数据库等方向研究的团队，隶属于数据工程与知识工程教育部重点实验室（DEKE）和信息学院计算机系。

该实验室主要包括海量 Web 数据管理、空间数据库管理技术、分布式与云计算以及 XML 数据查询和管理 4 个主要研究方向。研究内容包括海量数据管理的理论知识（一致性理论、分区策略、容错策略、存储和查询模型等）、流行的数据管理方法和已推出的众多数据管理系统、空间数据的表示和建模、存储与索引、查询处理、空间数据挖掘、XML 查询优化、XML 关键字查询、XML 查询改写以及 XML Twig 查询等。

1.3.2 产业界现状

1. 国外公司大数据研究现状

（1）谷歌

MapReduce 是 2004 年由谷歌提出的面向大数据集处理的编程模型，起初主要用作互联网数据的处理，如文档抓取、倒排索引的建立等。但由于其简单而强大的数据处理接口和对大规模并行执行、容错及负载均衡等实现细节的隐藏，该技术一经推出便迅速在机器学习、数据挖掘、数据分析等领域得到广泛应用。

继 MapReduce 之后，谷歌又推出了 Big Query 服务，能够通过使用类 SQL 查询语言在几秒钟内筛选数十亿行的数据。具体来说，BigQuery 允许用户上传超大规模数据，并直接对数据进行交互式分析；对于开发者来说，BigQuery 还提供了基于 Web 服务的编程接口，使得开发者可以利用谷歌的后台架构运行 SQL 语句，对超大规模的虚拟数据库进行操作。BigQuery 引擎可以快速扫描 70TB 未经压缩处理的数据，并且可马上得到分析结果。从技术的角度看，BigQuery 是一个在云端的 SQL 服务，可以提供海量数据的实时分析，客户端不需要做任何事情。

（2）IBM

针对大数据问题，IBM 推出了 InfoSphere 大数据分析平台。该平台包括 BigInsights 和 Streams 两个产品系列，二者互补。BigInsights 对大规模的静态数据进行分析，它提供多节点的分布式计算，可以随时增加节点，提升数据处理能力。Streams 则采用内存计算方式分析实时数据。InfoSphere 大数据分析平台还集成了数据仓库、数据库、数据集成、业务流程管理等组件。

BigInsights 基于 Hadoop，增加了文本分析、统计决策工具，同时在可靠性、安全性、易用性、管理性等方面提供了相应工具，可与 DB2、Netezza 等系统集成，适合企业级应用需求。Streams 是一款满足即时处理、过滤和分析流数据需要的应用程序。需要注意的是，BigInsights 和 Streams 是数据仓库的补充，而不能直接代替数据仓库。一方面因为 Hadoop 等技术的成熟度较低，还需要进一步稳定，另一方面的原因是某些特定的企业应用需求还需要数据仓库的支持。

具体来说，BigInsights 静态大数据分析平台能够在常用、低成本的硬件上运行，并行支持线性可伸缩性，可用于支持半结构化或非结构化的信息，同时不需要烦琐的预处理，允许跨信息类型动态添加结构和关联。另外，它还可以支持主动风险管理与预测、实体识别与情绪趋势分析等新型工作负载，同时配备了高级文本分析功能。Streams 大数据实时分析平台则是一个擅长处理流动数据的高性能计算平台。它允许用户开发的应用在信息从成千上万个实时源到达时便快速对其进行采集、分析和关联操作，及时捕捉并处理关键业务数据。目前，Streams 能够满足用户当前对反应时间和可扩展性的要求，并支持高容量、结构化和非结构化流数据源。

（3）微软

微软在数据检索、数据处理和数据存储等方面对大数据问题进行了研究，开发出了一系列产品。

在数据检索方面，为了呈递高质量的搜索结果，微软在 Bing 中分析了超过 100PB 的数据。在数据存储方面，微软提出并行数据仓库（PDW）概念，能够处理超过 600TB 的大数据量，并提供企业级的计算能力。在数据处理与计算方面，微软为 LINQ to HPC（高性能计算）提供了分布式的运行时和编程模型，并支持将 Windows Server 和 Windows Azure 等平台构建在分布式的 Apache Hadoop 之上，以提高系统的处理能力和扩展性。

（4）SAS

自 1976 年以来，SAS 就一直致力于向企业提供数据分析服务，目前支持着世界上最大的数据集。SAS 的大数据产品主要包括高性能分析服务器（SAS High-Performance Analytics Server）、SAS 可视化分析（SAS Visual Analytics）和 SAS DataFlux 数据流处理引擎（SAS DataFlux Event Stream Processing Engine）。为科学计算、时间序列趋势预测、作业成本管理、金融大数据整体解决方案、客户智能、财务智能、政府行业解决方案等提供了有效的支持。

（5）EMC

EMC 针对大数据推出了 Greenplum 数据引擎软件，为新一代数据仓库所需的大规模数据和复杂查询功能提供支持。Greenplum 基于 MPP（海量并行处理）和 Shared-Nothing（完全无共享）架构，采用开源软件和 X86 商用架构。Greenplum 在其数据库中引入了 MapReduce 处理功能，其

执行引擎可以同时处理 SQL 查询和 MapReduce 任务，这种混合方式在代码级整合了 SQL 和 MapReduce：SQL 可以直接使用 MapReduce 任务的输出，同时 MapReduce 任务也可以使用 SQL 的查询结果作为输入。

针对 Hadoop，EMC 还推出了 GreenplumHD。该工具包含 Hadoop 分布式文件系统 HDFS、MapReduce、Hive、Pig、HBase 和 Zookeeper。GreenplumHD 包装了 Hadoop 的分布式技术，消除了从头开始构建分布 Hadoop 集群所带来的不便。Greenplum 也纳入到 Hadoop 的可插拔存储层，使用者能够在数据存储过程中选择多种存储方式而无需改变现有应用程序。

针对数据处理过程的协作问题，EMC 推出用于大数据处理的社交工具集 Greenplum Chorus，使得数据科学家可以通过类似 Facebook 的社交方式进行协作完成数据处理任务。该软件基于开放架构，能够用于数据挖掘和协作分析，包括数据探索、个人项目工作空间、数据协作分析和发布等几个主要环节。在数据探索阶段，Greenplum Chorus 通过搜索引擎快速查找数据，并将数据进行关联，从而实现数据采集的可视化；在处理阶段，采集来的数据被放到个人沙盒中进行处理，这个处理过程不会影响整个数据库的运行；在协作分析阶段，数据分析人员可以共享工作空间、代码，协同工作兼具灵活性和安全性；最后，相关的处理结果被发布出来。上述处理过程循环往复，最终完成数据处理工作。

（6）Teradata

Teradata 针对大数据问题，推出了 Aster Data 产品，该产品将 SQL 和 MapReduce 进行结合，针对大数据分析提出了 SQL/MapReduce 框架，该框架允许用户使用 C++、Java、Python 等语言编写 MapReduce 函数，编写的函数可以作为一个子查询在 SQL 中使用，从而同时获得 SQL 的易用性和 MapReduce 的开放性。除此以外，Aster Data 基于 MapReduce 实现了 30 多个统计软件包，从而将数据分析推向数据库内进行（数据库内分析），提高了数据分析的性能。

2. 国内公司大数据研究现状

（1）百度

百度作为最大的中文搜索引擎公司，拥有海量的数据，当前估计有三千亿左右的中文网页，大约有 10 至 50 个 PB，并且这些数据每隔一小时就会发生较大的变化。另外还拥有结构化的日志信息、高要求广告信息、百度知道、百度文库等用户实时产生的内容等。百度大数据的特点是大而杂，为了实现数据的实时性、一致性、可扩展性等高标准要求，百度采用了自行开发的存储系统，该系统有三个方面的特点：

- 网页存储，通过先存后写的策略将随机写过程转换成顺序写；
- 存储优化，包括针对访问模式的优化和单机性能的提升，等等；
- 删除 Flash Special，直接针对 Flash 多通道存储数据；利用多副本存储，服务器可以找到备份，保持业务的连续性；针对大文件进行拆片存储。

（2）阿里数据

2005 年，淘宝成立商业智能部门，开发了第一款数据分析产品——“淘数据”，为各业务公

司、部门提供经营报表。

2009 年，阿里数据开始进入产品化时代。“淘数据”从一个内部报表系统跃升为内部数据统称。2009 年 4 月和 12 月，商业智能团队又分别开发出可预警的“KPI 系统”、服务于业务部门的“数据门户”。

2009 年，将集团各公司自行搭建的 Hadoop 集群统一，开发出“云梯”系统，以实现公司内部所有数据的打通、整合的管理和共享。2010 年初，淘宝推出“数据魔方”，向市场开放全局市场数据。

2011 年，淘宝接连推出“观星台”、“地动仪”、“黄金策”、“淘宝指数”和“淘宝时光机”等多款大数据产品。“观星台”是一个高度可视化的仪表盘，选择最关键的数据在几秒内展示全局运营状况；“地动仪”则可以看到用户投诉最多的功能有哪些，甚至可以获取最原始的客服电话录音；“淘宝指数”可以告诉用户数据的长期走势、购买商品的人群特征、商品成交排行等重要信息。

（3）新浪

2013 年，新浪推出大数据产品——微博 Page，这是一个聚合了用户兴趣爱好、社交关系数据的综合展示页面，话题、图书、音乐、餐饮美食等内容都能在微博上生成专属的 Page 页面。通过 Page 页面，网友可以很方便地查看到有价值的微博内容。

2012 年 4 月 15 日，中国数据库技术大会（DTCC）“NoSQL 数据库创新专场”中新浪微博开放平台资深工程师唐福林发表主题演讲《新浪微博：Redis 的大数据之路》，介绍了 NoSQL 数据库 Redis 在新浪微博的使用场景及经验教训。新浪微博从 2010 年底开始使用 Redis，各项业务指标在经历了 2011 年全年的疯狂增长之后，发现在很多场合 Redis 已经不再适用。Redis 适用于数据量不太大的存储，以及数据量大的缓存。在选择数据存储介质的时候要分清数据量的大小和数据的冷热：小而热的数据适合使用内存，大而冷的数据适合使用磁盘，大而热的数据是否适合使用 SSD，仍待探讨。

（4）腾讯

腾讯的产品线非常广泛，从门户网站到微博、视频、电子商务、无线、开放平台等多个跨平台领域。腾讯的大数据战略，主要分为 2C（个人）和 2B（商家）两个部分，前者是提升用户体验，后者带来有效的广告收益。

腾讯将调动 7 亿活跃账户的数据支持门户服务，打造基于用户社交关系链的“下一代腾讯网”。下一代腾讯网利用大数据和关系链，为用户筛选、推荐最适合他的内容。在此基础上，腾讯的广告产品也将不再只是基于传统网络媒体的展示，而是更多基于用户社交关系链的口碑营销。

1.3.3　政府机构现状

1. 联合国大数据研究现状

联合国于 2012 年 7 月在纽约总部发布了一份大数据政务白皮书，总结了各国政府如何利用大数据更好地服务和保护人民。

在名为《大数据促发展：挑战与机遇》的白皮书中，联合国指出大数据对于联合国和各国政府来说是一个历史性的机遇，主要探讨如何利用包括社交网络在内的大数据资源造福人类。该报告是联合国“全球脉搏”项目的产物。“全球脉搏”是联合国发起的一个全新项目，旨在利用消费互联网的数据推动全球发展。利用自然语言解码软件，对社交网络和手机短信中的信息进行情绪分析，从而对失业率增加、区域性开支降低或疾病暴发等进行预测。

联合国的大数据白皮书还建议联合国成员国建设“脉搏实验室（Pulse Labs）”，开发网络大数据的潜在价值。印度尼西亚和乌干达作为两个标杆国家率先在各自的首都雅加达和坎贝拉建设了脉搏实验室。其中雅加达的脉搏实验室于 2012 年 9 月投入运行，由澳大利亚提供资助。

2. 美国政府大数据研究现状

2012 年 3 月 29 日美国政府公布了“大数据研发计划”（Big Data Research and Development Initiative）。该计划的目标是改进现有人们从海量和复杂的数据中获取知识的能力，从而加速美国在科学与工程领域发明的步伐，增强国家安全，转变现有的教学和学习方式。2012 年 3 月底，美国政府发布《大数据研究开发倡议》，以美国科学与技术政策办公室为首，美国国家科学基金会、美国国立卫生研究院、国防部、能源部等已经开始了与民间企业或大学开展多项大数据相关的各种研究开发。美国政府为此拨出超过 2 亿美元的研究开发预算。

在这一背景下，美国政府各个部门纷纷开展了相关的研究计划。

（1）**多尺度异常检测（ADAMS）项目。**该项目旨在解决大规模数据集的异常检测和特征化问题。项目中对异常数据的检测指对现实世界环境中各种可操作的信息数据及线索的收集。最初的 ADAMS 应用程序进行内部威胁检测，在日常网络活动环境中，检测单独的异常行动。

（2）**网络内部威胁（CINDER）计划。**该项目旨在开发新的方法来检测军事计算机网络与网络间谍活动。作为一种揭露隐藏操作的手段，CINDER 适用于将对不同类型对手的活动统一成“规范”的内部网络活动，并提高对网络威胁检测的准确性和速度。

（3）**Insight 计划**。该计划主要解决目前情报、监视和侦察系统的不足，进行自动化和人机集成推理，使得能够提前对时间敏感的更大潜在威胁进行分析。该计划旨在开发出资源管理系统，通过分析图像和非图像的传感器信息和其他来源的信息，进行网络威胁的自动识别和非常规的战争行为。

（4）**Machine Reading 项目**。该项目旨在实现人工智能的应用和发展学习系统的过程中对自然文本进行知识插入，而不是依靠昂贵和费时的知识表示目前的进程，并需要专家和相关知识工程师所给出的语义表示信息。

（5）**Mind's Eye 项目**。该项目旨在为机器建立视觉的智能。传统的机器视觉研究的对象选取广泛的物体来描述一个场景的属性名词，而 Mind's Eye 旨在增加在这些场景的动作认识和推理需要的知觉认知基础。总之，这些技术可以建立一个更完整的视觉智能效果。

（6）**视频和图像的检索和分析工具（VIRAT）计划。**该计划旨在开发一个系统，能够利用图像分析师收集的数据进行大规模军事图像分析。VIRAT 希望能够帮助图像分析师在相关活动发生时建立警报。该系统还包含一套开发工具，能够以较高的准确率和召回率从大量视频库中对视

频内容进行检索。

（7）**XDATA 项目**。该项目旨在开发用于分析大量半结构化和非结构化数据的计算方法和软件工具。该项目需要解决的核心问题包括：可伸缩算法在分布式数据存储环境中的应用方式；如何使人机交互工具有效、迅速定制不同任务，以方便对不同数据进行可视化处理；灵活使用开源软件工具包，使得能够处理大量国防应用中的数据，等等。

（8）**Mission-oriented Resilient Clouds 项目**。该项目通过云计算技术进行检测，诊断并对大量攻击行为做出响应；建立“社区卫生服务云”，以解决云计算环境中的大量安全挑战。该项目还采用新技术，提高云计算基础设施环境和其中的大数据应用系统的可用性。保证系统受到攻击时，只要整体能够有效运行和保存，允许个别主机和任务失效。

（9）**对加密数据的编程计算（PROCEED）项目**。该项目希望研发一套实用的方法，支持用户采用高级编程语言，在不需要首次解密的情况下操纵已加密的数据，使得对手拦截信息更加困难，大大提高数据的安全性。

3. 日本政府大数据研究现状

在日本工业界，本田、先锋等企业推出的基于 GPS 的“道路通行图”在受灾地区救助活动中得到了应用展示；NTT DoCoMo 公司推出的基于匿名化的手机定位信息展现人口移动的“移动空间统计”也是一个大数据应用案例。但是，日本因为企业结构上的垂直性组织以及隐私保护等问题，往往难以有效采集信息，所以迫切需要建设大数据应用所需的平台。

在上述背景下，日本总务省于 2012 年 7 月推出 ICT 战略研究计划：“活力 ICT 日本”，该计划将重点关注“大数据应用”。该计划指出：“提升日本竞争力，大数据应用不可或缺。”该计划将“大数据”定义为从各种传感器、社会化媒体等处采集到的海量信息，经过数据分析，用于提升经济活动的效率。在委员会中担任大数据研究主任的东京大学教授森川博之强调，美国在技术上处于领先，日本也应将其定位为战略领域之一。

日本的 ICT 战略将重点关注大数据应用所需的云计算、传感器、社会化媒体等智能技术开发。新医疗技术开发、缓解交通拥堵等公共领域将会得到大数据带来的便利与贡献。根据日本野村综合研究所的分析显示，日本大数据应用带来的经济效益将超过 20 万亿日元。

4. 中国政府大数据应用现状

在大数据领域的落后，意味着国家安全将在数字空间出现漏洞，国家创新能力将在未来国际竞争中落后于人。因此，中国政府一直大力主导、加快推进大数据技术的研发性应用。

向服务型政府转变、政务公开、公共事业的发展等，信息社会的不断进步让政府部门越来越依赖数据的分析进行决策。对大数据进行深度挖掘、发展趋势分析，积极探索更多的应用场景和模式，是中国政府在 2013 年大数据主流应用方向。2012 年，国内已经有一些省市陆续启动了大数据战略，包括建设政务数据中心、成立大数据研究机构等，大数据正在成为全社会的战略资源。

从国家层面上看，国务院《“十二五”国家战略性新兴产业发展规划》中提出，海量数据存储、处理技术的研发与产业化将作为我国未来战略新兴产业的一个方面；工业和信息化部《物联网“十二五”发展规划》也将信息处理技术列为 4 项关键技术创新工程之一，其中包括海量数据存储、

数据挖掘、图像视频智能分析，另外 3 项关键技术创新工程，包括信息感知技术、信息传输技术、信息安全技术，也是大数据产业的重要组成部分，都与大数据产业的发展密不可分。在科学研究领域，2014 年度的“973”项目指南中，“大数据计算的基础研究”已经作为一项重点研究课题，主要面向网络信息空间大数据挖掘的需求，结合 1~2 种重要应用，研究多源异构大数据的表示、度量和语义理解方法，研究建模理论和计算模型，提出能效优化的分布存储和处理的硬件及软件系统架构，分析大数据的复杂性、可计算性与处理效率的关系，为建立大数据的科学体系提供理论依据。

从地方政府层面上看，许多城市都在开展与智慧城市相关的信息化建设项目，包含测绘以及城市地图、政府管理与决策的信息化、企业管理决策与服务的信息化以及数字城市生活等方方面面。以智慧北京为例，近年来，北京各相关单位积极筹建各类信息化系统，在基础设施建成的基础上，逐步实现信息、数据共享，建成北京市现代城市交通网络系统、数字政务管理系统，以及北京旅游信息系统等。在实现部分系统城市内信息互通的基础上，与一些地区逐步实现联网，达到资源远程共享。除此以外，政务信息开放、舆情监控、平安城市、预警预案等多种项目的实施也在不断推动大数据产业的蓬勃发展。

1.4 大数据的应用领域

全球著名咨询公司麦肯锡认为，只要给予适当的政策支持，“大数据”将引发新一轮的生产力增长与创新，“大数据”中蕴含着尚未开发的巨大价值。例如，充分利用“大数据”的零售商将能够使营业额利润率提高 60%以上；如果美国医疗保健行业有效利用“大数据”，就能够把成本降低 8%左右，从而每年创造出 3000 多亿美元的产值；在欧洲发达国家，如果政府利用“大数据”提高运作效率，那么将节省至少 1000 亿欧元的成本；而利用个人位置数据提供的服务将可以创造 6000 亿美元的消费者剩余。

大数据的迅速增长及相关技术的发展正在带来全新的商业机遇。2011 年完成的“新智能企业全球高管调查和研究项目”指出，绝大多数企业都已抓住了大数据的机遇：2011 年，58%的企业已经将大数据分析技术用于在市场或行业内创造竞争优势，而 2010 年这一比例仅为 37%。值得注意的是，采用分析技术的企业持续超越同行的可能性要高两倍。

麦肯锡对医疗保健、零售、公共领域、制造、个人位置数据这 5 大领域进行了重点分析，提出了可以利用“大数据”的 5 种方法。

（1）以时效性更高的方式向用户提供“大数据”。在公共领域，跨部门提供“大数据”能大幅减少检索与处理时间。在制造业，集成来自研发、工程、制造单元的数据可以实现并行工程，缩短产品投放市场的时间。

（2）通过展开数据分析和实验寻找变化因素并改善产品性能。由于越来越多的交易数据都以数字形式存在，各机构可以收集有关产品或用户的更加精确和详尽的数据。

（3）区分用户群，提供个性化服务。“大数据”能帮助机构对用户群进行更加细化的区分，并针对用户的不同需求提供更加个性化的服务。这是营销和危机管理方面常用的方法，但也可以为公共领域等带来变革。

（4）利用自动化算法支持或替代人工决策。复杂分析能极大改善决策效果，降低风险，并挖掘出其他方法无法发现的宝贵信息。此类复杂分析可用于税务机构、零售商等。

（5）商业模式、产品与服务创新。制造商正在利用产品使用过程中获得的数据来改善下一代产品开发，以及提供创新性售后服务。实时位置数据的兴起带来了一系列基于位置的移动服务，例如导航和人物跟踪。

通过上述分析，可以看出大数据可在制造业、服务业、交通、医疗等领域得到广泛的应用，下面将一一进行阐述。

1.4.1　大数据在制造业的应用

制造业目前正在向信息化和自动化的方向发展。在产品的设计、生产和销售中，越来越多的企业使用计算机辅助设计（CAD）、计算机辅助制造（CAM）等软件，数控机床、传感器等设备，物料需求计划（MRP）、企业资源计划（ERP）等系统。这些信息技术的应用大大提高了工作效率和产品质量。

然而，随着信息化的不断深入，制造业目前所面临的挑战是在产业化和信息化之后，如何提升获取和开拓市场需求的能力，从而创造出更有价值的商品。如今，企业管理信息系统中存储的信息，各种工业传感器和数控设备中产生的数据，都将汇集到一起形成大数据，以提高生产效率为目标的信息化制造业转变成以掌握用户需求为目标的智慧化制造业。大数据为制造业的创新转型（无论是精益化提升还是服务化转型）提供了新的路径和方式。

另一方面，海量数据扩大了算法和运筹学的应用领域。例如，在部分制造企业，算法对生产线的传感器信息进行分析，形成了自我调节的流程，从而减少了浪费，避免了代价高昂（有时还十分危险）的人为干预，最终提升产量。在先进的“数码化”油田，仪表不时读取有关井口状况、管道和机械系统的各类数据，这些信息由一组计算机进行分析，并将结果输入实时运营中心。运营中心则调整油量以优化生产和最大限度缩短停机时间。基于这一思路，一家大型石油公司减少了 10%~25%的运营成本和员工成本，产量则提高了 5%。

现在，从复印机到喷气发动机等各种产品都可以产生能跟踪其使用情况的数据流。制造商能够分析输入数据，并有可能主动纠正软件缺陷或派遣服务代表到现场维修。一些计算机硬件供应商正在收集和分析这些信息，在发生故障导致客户运营中断前未雨绸缪，提前维护。这些信息还可以用于实施产品变化、预防未来问题的发生、提供客户使用信息等方面，为下一代产品开发提供灵感和思路。

1.4.2 大数据在服务业的应用

传统的服务业有着悠久的历史。当信息时代到来的时候，服务业就衍化出现了两种形态：一种是信息技术与服务业相结合的信息服务业，另一种是应用信息技术改造传统服务业而来的服务业。前者包括计算机软件、通信服务、信息咨询服务等，后者包括信息化改造后的商业、金融业、旅游业等。大数据恰恰就在这两者之间起到牵线搭桥的作用：一方面它使得信息服务业从提供软硬件技术服务升级到提供智慧解决方案，另一方面它将改变现有的服务业业态模式，将关注点转向数据。

在信息服务业，最常见的大数据分析当属网络公司收集用户的网页单击行为提供有个性化的广告与信息推送服务，需要注意的是这些行为需要考虑用户隐私的保护问题。

在信息化改造后的服务业，大数据更是无处不在。在零售行业，厂商可以通过互联网单击流实时跟踪客户行为、更新客户偏好、建立可能行为的模型。在此基础上，厂商能够确定客户下次购买的时间，通过捆绑优选商品、提供省钱的奖励性计划、对交易实施微调等措施，最终使得整个销售圆满结束。在金融行业，银行可以从大量数据中发现信用卡欺诈和盗用；理财网站从统计的消费数据中来预测宏观的经济趋势；保险公司通过大数据能够找出可疑的权利要求。在旅游行业，企业致力于旅游预订数据的收集、分析与处理，例如微软的 Bing 搜索引擎能够根据其存储的机票历史数据，帮助用户决定购买航班的最佳时间和最优惠价格。

1.4.3 大数据在交通行业的应用

当前，出行难问题对各大城市来说都亟待解决。可以利用先进的传感技术、网络技术、计算技术、控制技术、智能技术，对道路和交通进行全面感知。而在大数据时代下的智慧交通，需要融合传感器、监视视频和 GPS 等设备产生的海量数据，甚至与气象监测设备产生的天气状况等数据相结合，从中提取出人们真正需要的信息，及时而准确地进行发布和传送，通过计算直接提供最佳的出行方式和路线。

1.4.4 大数据在医疗行业的应用

医疗保健问题是当前社会普遍关注的焦点问题。以往，人们总是在发现自己生病时才看病就医，而且到了医院还要经历挂号、求诊、配药等复杂流程，整个过程需要耗费大量时间，容易形成就医难的困境。如今，基于电子医疗记录技术，电子病历正逐渐被各大医疗机构所采用。在去医院前，可以通过网上预约挂号；在就医时，仅使用一张 IC 卡就能付费；医生还可以将问诊过程中的记录，病人的化验单、拍片等诊断数据输入电脑以备随时调用。

在大数据时代，可以将医疗机构的电子病历记录标准化，形成全方位多维度的大数据仓库。系统首先全面分析患者的基本资料、诊断结果、处方、医疗保险情况和付款记录等诸多数据，再将这些不同的数据综合起来，在医生的参与下通过决策支持系统选择最佳的医疗护理解决方案。

1.5 本章小结

本章首先对大数据产生过程的相关历史进行了介绍，并对大数据的 5 个代表性定义进行了解读，帮助读者对“大数据”这一概念进行深入理解。本章的第二部分以 ACM 数据库专家组的观点为例，介绍了大数据的主要研究内容，并总结了大数据在国内外产业界、学术界和政府机构的研究现状。最后，本章介绍了大数据的可能应用领域，并重点阐述了具有代表性的制造、服务、交通和医疗 4 个行业的实际应用现状与前景。希望读者通过本章的学习，能够掌握大数据的基本概念、发展历史、研究内容和研究现状。

第 2 章

数据存储技术

本章主要介绍了数据存储技术的概念与研究现状，分析了海量数据存储的关键技术，最后介绍了海量数据存储的实现与工具。通过本章的学习，读者可对数据存储有充分了解。

随着经济全球化的不断发展，国际性的大型企业不断涌现，其数以亿万计的用户来自全球各地，其业务数据量非常大。另外还有一些传统的大数据量应用单位，如国家地震局、国家气象局、国家图书馆、中央电视台等其数据量可多达几十 PB。对于这样的单位和企业，如何解决其大量数据存储与有效使用成为其业务的核心问题之一。其对数据存储提出了诸多要求：

- 对性能的要求。由于这些大型数据中心将承载全国乃至全球巨大的检索访问量，所以对后台存放检索信息数据的存储设备性能要求极高。数字资源的全球共享使得数据交换量激增，同样也要求后台存储设备提供极高的性能。
- 对容量的要求。地震局、气象局拥有数亿万份遥感观测数据、中间计算结果、历史数据等；图书馆拥有数千万的电子图书、电子刊物、电子图片等；电视台拥有数百万份音频资料、视频资料等；大型企业拥有数亿份的客户资料。每一家的数据都将占用数十 PB 的存储容量，而随着数据资源的不断丰富和交换，容量还将不断扩大。
- 对数据资源有效管理的要求。当拥有海量数据后就要对海量数据进行有效的管理，合理利用 IT 软件、硬件设施管理这些数据，使得数据在其生命周期内得到最大的利用率，而消耗最少的 IT 资源，以得到最大的投资回报。
- 对数据资源保护的要求。大型数据中心内的海量数据不是生死攸关的用户数据，就是凝结了无数工程人员心血的劳动成果，这些数据资源都是企业和单位最宝贵的财富，对于这些宝贵的财富应该采取一定的措施避免由于人为误操作、设备损坏、火灾等意外情况造成的损失。

综上所述，海量数据存储的解决方案至关重要，本章将介绍数据获取与存储技术的概念以及一些流行的工具与实现。

2.1 数据存储技术介绍

随着大数据应用的爆发性增长，它已经衍生出了自己独特的架构，而且也直接推动了存储、网络以及计算技术的发展。毕竟处理大数据这种特殊的需求是一个新的挑战。硬件的发展最终还是由软件需求推动的，可以很明显地看到大数据分析应用需求正在影响着数据存储基础设施的发展。从另一方面看，这一变化对存储厂商和其他 IT 基础设施厂商未尝不是一个机会。随着结构化数据和非结构化数据量的持续增长，以及分析数据来源的多样化，此前存储系统的设计已经无法满足大数据应用的需要。存储厂商已经意识到这一点，开始修改基于块和文件的存储系统的架构设计以适应这些新的要求。

针对大数据的世界领先品牌存储企业有：IBM、EMC、LSISandForce 、 INTEL、惠普、戴尔、甲骨文、日立、赛门铁克等。

对于大数据的存储，存在以下几个不可忽视的问题。

1. 容量问题

这里所说的“大容量”通常可达到 PB 级的数据规模，因此，海量数据存储系统也一定要有相应等级的扩展能力。与此同时，存储系统的扩展一定要简便，可以通过增加模块或磁盘柜来增加容量，甚至不需要停机。在解决容量问题上，不得不提及 LSI 公司的全新 Nytro 智能化闪存解决方案，采用 Nytro 产品，客户可以将数据库事务处理性能提高 30 倍，并且超过每秒 4.0GB 的持续吞吐能力，非常适用于大数据分析。

2. 延迟问题

“大数据”应用还存在实时性的问题。特别是涉及与网上交易或者金融类相关的应用时。有很多“大数据”应用环境需要较高的 IOPS 性能，比如 HPC 高性能计算。此外，服务器虚拟化的普及也导致了对高 IOPS 的需求，正如它改变了传统 IT 环境一样。为了迎接这些挑战，各种模式的固态存储设备应运而生，小到简单的在服务器内部做高速缓存，大到全固态介质可扩展存储系统通过高性能闪存存储，自动、智能地对热点数据进行读/写，高速缓存的 LSI Nytro 系列产品等都在蓬勃发展。

3. 安全问题

某些特殊行业的应用，比如金融数据、医疗信息以及政府情报等都有自己的安全标准和保密性需求。虽然对于 IT 管理者来说这些并没有什么不同，而且都是必须遵从的，但是，大数据分析往往需要多类数据相互参考，而在过去并不会有这种数据混合访问的情况，大数据应用催生出一些新的、需要考虑的安全性问题，这就充分体现出利用基于 DuraClass™ 技术的 LSI SandForceR 闪存处理器的优势，实现了企业级闪存性能和可靠性，实现简单、透明的应用加速，既安全又方便。

4. 成本问题

对于那些正在使用大数据环境的企业来说，成本控制是关键的问题。想控制成本，就意味着要让每一台设备都实现更高的“效率”，同时还要减少那些昂贵的部件。目前，像重复数据删除等技术已经进入主存储市场，而且现在还可以处理更多的数据类型，这都可以为大数据存储应用带来更多的价值，提升存储效率。在数据量不断增长的环境中，通过减少后端存储的消耗，哪怕只是降低几个百分点，这种锱铢必较的服务器（也只有 LSI 推出的 Syncro™ MX-B 机架服务器启动盘设备）都能够获得明显的投资回报，当今，数据中心使用的传统引导驱动器不仅故障率高，而且具有较高的维修和更换成本。如果用它替换数据中心的独立服务器引导驱动器，则能将可靠性提升多达 100 倍。并且对主机系统是透明的，能为每一个附加服务器提供唯一的引导镜像，可简化系统管理，提升可靠性，并且节电率高达 60%，真正做到了节省成本。

5. 数据的积累

许多大数据应用都会涉及法规遵从问题，这些法规通常要求数据要保存几年或者几十年。比如医疗信息通常是为了保证患者的生命安全，而财务信息通常要保存 7 年。而有些使用大数据存储的用户却希望数据能够保存更长的时间，因为任何数据都是历史记录的一部分，而且数据的分析大都是基于时间段进行的。要实现长期的数据保存，就要求存储厂商开发出能够持续进行数据一致性检测的功能以及其他保证长期高可用的特性。同时还要实现数据直接在原位更新的功能需求。

6. 灵活性

大数据存储系统的基础设施规模通常都很大，因此必须经过仔细设计，才能保证存储系统的灵活性，使其能够随着应用分析软件一起扩容及扩展。在大数据存储环境中，已经没有必要再做数据迁移了，因为数据会同时保存在多个部署站点。一个大型的数据存储基础设施一旦开始投入使用，就很难再调整了，因此它必须能够适应各种不同的应用类型和数据场景。

7. 应用感知

最早一批使用大数据的用户已经开发出了一些针对应用的定制的基础设施，比如针对政府项目开发的系统，还有大型互联网服务商创造的专用服务器等。在主流存储系统领域，应用感知技术的使用越来越普遍，它也是改善系统效率和性能的重要手段，所以，应用感知技术也应该用在大数据存储环境里。

8. 针对小用户

依赖大数据的不仅仅是那些特殊的大型用户群体，作为一种商业需求，小型企业未来也一定会应用到大数据。有些存储厂商已经在开发一些小型的“大数据”存储系统，主要吸引那些对成本比较敏感的用户。

2.2 数据采集与存储技术研究现状

目前，大部分互联网应用仍然使用传统关系型数据库进行数据的存储管理，并通过编写 SQL 语句或者 MPI 程序来完成对数据的分析处理。这样的系统在用户规模和数据规模都相对较小的情况下，可以高效地运行。但是，随着用户数量、存储管理的数据量的不断增加，许多热门的互联网应用在扩展存储系统以应对更大规模的数据量和满足更高的访问量时都遇到了问题。

2.2.1　传统关系型数据库

传统关系型数据库在数据存储管理的发展史上是一个重要的里程碑。在互联网时代以前，数据的存储管理应用主要集中在金融、证券等商务领域。这类应用主要面向结构化数据，聚焦于便捷的数据查询分析能力、按照严格规则快速处理事物的能力、多用户并发访问能力以及数据安全性的保证。而传统关系型数据库正是针对这种需求而设计的，并以其结构化的数据组织形式、严格的一致性模型、简单便捷的查询语言、强大的数据分析能力以及较高的程序与数据独立性等优点获得广泛应用。

然而随着互联网时代的到来，数据已超出关系型数据库管理的范畴，电子邮件、超文本、博客、标签以及图片、多媒体等各种非结构化数据逐渐成为了需要存储和处理的海量数据的重要组成部分。面向结构化数据存储的关系型数据库已经不能满足互联网数据快速访问、大规模数据分析的需求。主要表现在以下几个方面。

1. 应用场景的局限性

传统数据库在设计上，着眼于面向结构化的数据，致力于事务处理，要求保证严格的一致性。这些特性符合传统的金融、经济等应用场景。然而互联网应用主要面向半结构化、非结构化的数据，这些应用大多没有事务特性，也不需要很严格的一致性保证。虽然传统数据库的厂商也针对海量数据应用特点提出了一系列改进方案，但是由于并不是从互联网应用的角度去设计解决方案，使得传统数据库在应对互联网海量数据存储效果上并不理想。

2. 关系模型束缚对海量数据的快速访问能力

关系模型是一种按内容访问的模型。即在传统的关系型数据库中，根据列的值来定位相应的行。这种访问模型，会在数据访问过程中引入耗时的输入输出，从而影响快速访问的能力。虽然，传统的数据库系统可以通过分区的技术（水平分区和垂直分区），来减少查询过程中数据输入输出的次数以缩减响应时间，提高数据处理能力，但是在海量数据的规模下，这种分区所带来的性能改善并不显著。

关系模型中规格化的范式设计与 Web 2.0 的很多特性相互矛盾。以标签为例，标签的分类模型是一种复杂的多对多关系模型。传统数据库的范式设计要求消除冗余性，因此标签和内容将会被存储在不同的表中，导致对于标签的操作需要跨表完成（在分区的情况下，可能需要跨磁盘、

跨机器操作），致使系统性能低下。

3. 缺乏对非结构化数据的处理能力

传统的关系型数据库对数据的处理只局限于某些数据类型，比如数字、字符、字符串等，对非结构化数据（图片、音频等）的支持较差。然而随着用户应用需求的提高、硬件技术的发展和互联网上多媒体交流方式的推广，用户对多媒体处理的要求从简单的存储上升为识别、检索和深入加工，面对日益增长的处理庞大的声音、图像、视频、E-mail 等复杂数据类型的需求，传统数据库已显得力不从心。

4. 扩展性差

在海量规模下，传统数据库的一个致命弱点，就是其可扩展性差。解决数据库扩展性问题，通常有两种方式：向上扩展（Scale up）和向外扩展（Scale out）。这两种扩展方式分别从两个不同的维度来解决数据库在海量数据下的压力问题。向上扩展，简而言之就是通过硬件升级，提升速度来缓解压力问题；而向外扩展则是通过将海量数据按照一定的规则进行划分，将原来集中存储的数据分散到不同的物理数据库服务器上。Sharding 正是在向外扩展的理念指导下，为传统数据库提出的一种解决扩展性的方案。Sharding 通过叠加相对廉价设备的方式实现存储和计算能力的扩展，其主要目的是为突破单节点数据库服务器的输入输出能力限制，提高快速访问能力，以及提供更大的读写带宽。但是，在互联网的应用场景下，这种解决扩展性的方案仍然存在着一定局限性。比如，数据存储在多个节点，需要考虑负载均衡的问题，这就要求互联应用实现复杂的负载自动平衡机制，引入较高代价；数据库严格的范式规定，使得表示成关系模型的数据很难划分到不同的 shard 中；同时，还存在一些数据可靠性和可用性的问题。

2.2.2 新兴数据存储系统

在传统关系型数据库已不能满足互联网应用需求的情况下，开始出现一些针对结构化、半结构化，甚至非结构化数据的管理系统。在这些系统中，数据通常采用多副本的方式进行存储，以保障系统的可用性和并发性；采用较弱的一致性模型（如最终一致性模型），在保证低延时的用户响应的同时，维持副本之间的一致状态；并且这些系统都提供良好的负载平衡策略和容错手段。

按照存储管理方式划分，这些新兴的数据存储管理系统可以归为两大类。

1. 集中式数据存储管理系统

这类系统采用传统的服务器群架构。整个系统需要一个主控节点维护各从节点的元信息，是一种集中控制的管理手段。其优势在于，集中管理的方式人为可控且维护方便，在处理数据同步时更为简单。其劣势在于，系统存在单点故障的危险。这类系统包括谷歌的 Bigtable 和雅虎的 PNUTS。

Bigtable 是谷歌开发的一套结构化存储系统。数据以多维顺序表的方式进行存储。整个系统采用传统的服务器群形式，由一个主控服务器和多个子表服务器构成，并使用分布式锁服务

Chubby 进行容错等管理。

PNUTS 是雅虎内部使用的，用于跨数据中心进行部署的大规模并行数据管理系统。它在数据中心内部采用与 Bigtable 类似的集中式管理体系。PNUTS 支持以顺序表和哈希表两种方式进行结构化数据的组织存储，并通过一定的优化手段在保证用户低延时访问服务的同时，提高数据批量载入的性能。

2. 非集中式数据存储管理系统

在这类系统中，各节点无主从之分，各节点通过相应的通信机制相互感知，自我管理性较强。其优势在于：由于没有主控节点，因而避免单点失效带来的危险；不需要过多人工干预。其劣势在于：由于无主控节点因而一些元数据更新操作的实现较为复杂；不易进行人工控制。亚马逊（Amazon）的 Dynamo 和 Facebook 的 Cassandra 即采用这种方式。

Dynamo 是一个基于分布式哈希的去中心化大规模数据管理系统。在 Dynamo 中，数据按照键/值对（key-value）进行组织，主要面向原始数据的存储。这种架构下，系统中每个节点都能相互感知，自我管理性能较强，没有单点失效。Cassandra 是 Facebook 开发的一套采用对等网络计算（Peer to Peer，P2P）技术实现的结构化数据存储系统。与 Dynamo 有所不同的是，Cassandra 采用类似 Bigtable 的多维表数据模型组织数据。

2.3 海量数据存储的关键技术分析

扩展性是互联网应用需求下海量数据存储的首要问题。构建一个 TB 级甚至 PB 级的数据存储系统，需要有自适应的数据划分方式、良好的负载均衡策略来满足数据、用户规模的不断增长需求。同时，在保证系统可靠性的同时，需要权衡数据一致性与数据可用性，来满足互联网应用低延时、高吞吐率的要求。在这一节中，主要从数据划分、数据一致性与可用性、负载均衡、容错机制 4 个主要方面来讨论构建一个高可靠、可扩展的海量数据存储系统的关键问题和技术。

2.3.1　数据划分

在分布式环境下，数据存储需要跨越多个存储单元。如何进行数据的划分是影响扩展性、负载平衡以及系统性能的关键问题。为了提供低延时的系统响应，克服系统性能的瓶颈，系统必须在用户请求到来时将请求进行合理分发。现有的海量数据管理系统主要采用哈希映射和顺序分裂这两种方式。在互联网应用中，数据通常以键/值对方式进行组织以适应数据的多样性和处理的灵活性。哈希映射是根据数据记录的键值进行哈希，根据哈希值将数据记录映射到相应的存储单元。但是这种数据划分方式带来的性能收益依赖于哈希算法的优劣。而顺序分裂则是一种渐进式的数据划分方式。数据按键值排序写入数据表中，数据表在其大小达到阈值后进行分裂，分裂后的数据将被分配到不同的节点上去继续提供服务。这样，新流入的数据根据键值自动找到相应的分片

插入表中。

Dynamo 和 Cassandra 都采用了一致性哈希的方式进行数据划分。这种方式在数据流入时就将数据均匀地映射到相应的存储单元，从而最大限度地避免系统热点的产生。同时一致性哈希算法也为系统带来了良好的扩展性。

而 Bigtable 则使用顺序分裂的方式进行数据划分。这种渐进式的数据划分方式，可以有效利用系统资源，并能提供很好的扩展性。但是某个键值范围的频繁插入可能产生负载热点。与哈希方式不同的是，顺序分裂的数据与存储节点并不存在直接映射的关系，在 Bigtable 中需要有一个主控节点来集中管理这种分裂和映射行为。因此，整个系统的扩展性最终受限于主控节点的管理能力。

虽然 PNUTS 提供了顺序表和哈希表两种数据的组织形式，但是其哈希表中的数据按照键的哈希值有序存放。就是说，PNUTS 采用了顺序分裂的方式来按照键或者键哈希值划分顺序表或者哈希表中的数据。

虽然这些系统采用不同的数据模型（键/值对、顺序表、哈希表等）来进行数据的组织，但是它们都根据这些数据组织的特性实现可扩展的数据划分方式。根据应用数据的特性，确定合理的数据划分策略以达到高可扩展性是海量数据存储系统设计的首要问题。

2.3.2 数据一致性与可用性

数据可用性是分布式环境下数据存储的基石，而数据一致性模型则为保证数据操作的正确性做出了限定。在分布式环境下，通常采用副本冗余、日志等方式来解决数据的可用性问题。但是副本冗余存储也带来了数据一致性的问题。在采用副本冗余方式的分布式系统中，数据一致性与系统性能是一对不可调和的矛盾，往往需要在系统的性能（如响应时间等）与数据的严格一致性之间进行折中。在低延时用户响应的互联网应用需求下，通常要牺牲数据严格的一致性来调和这种矛盾，即允许系统通过弱化一致性模型来保证高效的系统响应，同时通过异步复制的手段来保证数据的可用性。

Dynamo、Bigtable 和 PNUTS 都是通过副本冗余的方式来保证数据的高可用。但是，其具体实现又不尽相同。由于 Dynamo 采用非集中的管理方式，整个系统中无主从节点之分，因此是在整个哈希环上通过 gossip 机制进行通信来完成副本的异步复制。而采用集中管理方式的 Bigtable 和 PNUTS 均采用日志的方式保证服务节点内存中数据的可用性。不同的是，在数据存储可用性方面，BigTable 依赖于底层分布式文件系统的副本机制；而 PNUTS 则采用基于发行/订阅（pub/sub）通信机制的主从式异步复制方式来完成数据的冗余存储：数据首先被同步到主副本，然后通过发行/订阅机制异步更新到所有副本。

如上所述，需要跨数据中心部署的 Dynamo 和 PNUTS 都采用异步复制的方式进行副本更新，牺牲一定程度的数据一致性来保证系统的高性能。可见，数据一致性、可用性与系统性能的权衡考虑是一个与应用特性和部署方式紧密相关的问题。

2.3.3　负载均衡

负载均衡是分布式环境下进行高效数据管理的关键问题。它主要包括数据的均衡和访问压力的均衡这两个方面。在分布式环境中，数据通过一定的划分策略（哈希或者顺序分裂等）进行划分并存储在不同的节点上，用户的访问请求也将由不同的节点处理。由于用户访问请求分布规律的不可预测性导致最终数据存储分布的不均衡，以及节点访问压力的不均衡。在数据分布、访问负载不均衡的情况下，频繁的并发访问和持续的数据加载压力将会影响整个系统的性能。为了保证数据加载的高吞吐率、系统响应的低延时以及系统的稳定性，海量存储系统需要有一套良好的均衡机制来解决上述问题。

Dynamo 采用了虚拟节点技术，通过虚拟化的手段将节点的服务能力单元化，将访问压力较大的虚拟节点映射到服务能力较强的物理节点，达到访问压力的均衡。访问压力的均衡同时伴随着数据的均衡。为了最小化数据均衡过程中数据迁移的开销，Dynamo 同样采用虚拟化技术，量化节点的存储能力，将虚拟后的存储节点相对均匀地分散到集群哈希环上，避免数据均衡过程中全环的数据移动。在非集中式系统中，这些均衡操作可以由任一节点发起，通过 gossip 通信机制与集群中的其他节点协调完成。

与 Dynamo 这种非集中式管理不同的是，BigTable 通过主控节点（master）来监控各个子表服务器（tablet server）上的访问负载状态，利用主控节点调度管理子表的分裂和迁移，将访问压力均匀地分散到各个子表服务器上。由于 BigTable 采用分布式文件系统作为数据的底层存储，因而访问压力均衡过程中并不涉及存储数据的迁移操作，以一种巧妙的方式避免了数据均衡的问题。在集中式管理系统中，PNUTS 也采用类似的方式进行访问压力的均衡。不同的是，采用本地文件系统或者本地数据库系统的 PNUTS 在进行子表（tablet）的分裂和迁移时，需要进行存储数据迁移。

由此可见，有效的数据划分方式为系统扩展性提供了一个基础，但是同时也给系统带来了负载均衡的问题。通过虚拟化节点或者表分裂等方式改变数据分布格局，均衡访问负载的同时，尽可能减少存储数据迁移量或者避免数据迁移，是海量存储系统的一个挑战。

2.3.4　容错机制

容错是分布式系统健壮性的标志。节点的失效侦测以及失效恢复已经成为保证系统的可用性、可靠性的关键问题。

1. 失效侦测

在非集中式系统中，各节点之间定期进行交互以了解节点的活动状态，从而侦测失效的存在，如 Dynamo、Cassandra。而在集中式系统中，整个系统需要有专门的部件（节点）来维护整个分布式系统中节点的状态信息，并通过“心跳”机制完成失效节点的侦测。如 Bigtable 通过分布式锁服务 chubby 来跟踪主控节点和子表节点的服务状态，完成节点的失效侦测；PNUTS 则利用子表控制器（tablet controller） 部件维护的活动节点路由信息来判断节点失效的存在。

2. 失效恢复

在系统侦测到失效节点的存在后，需要一定的恢复策略来完成对失效节点的恢复，保证系统的可用性和可靠性。在分布式系统中，节点的失效分为临时失效（如网络分区等）和永久失效（如节点死机、磁盘损坏等）两种情况。在副本冗余存储的分布式系统中，失效通常会造成多副本之间的数据不一致，需要对失效节点的数据进行同步来完成失效的恢复。同时，永久失效通常会造成失效节点内存中数据的丢失，日志重做通常是解决这类问题的一种办法。当然，具体的失效恢复策略在不同的系统中又各有特色。

以 BigTable 为例。临时失效和永久失效在 BigTable 中并不做区分。BigTable 依靠主控节点通过“心跳”机制来侦测失效的存在，即在规定时间内主控节点无法通过“心跳”获得从节点响应就认为该从节点失效。即使临时失效的节点可能再次与主控节点建立连接，这些节点也将被主控节点停止，因为这些节点上的服务已经被重新分配到其他节点上。服务的迁移并不涉及存储数据的移动，不会引入额外的系统开销，因而也就无需区分各种失效状态。这种依赖于底层分布式文件系统的共享存储方式，简化了系统的失效恢复。

在集中式系统中，主从节点的功能差异使得主节点各种失效恢复的方式不尽相同。由于主节点维护系统元信息，因此其失效将是灾难性的。在集中式系统中，通常采用备份节点（双机、多机备份）来防止主节点失效的发生。然而 Bigtable 则通过 chubby 来管理集群节点的状态信息，利用子表服务器来管理整个系统的存储元信息，弱化主节点的管理功能，减小主节点失效导致灾难的可能性，同时也降低了主节点恢复的复杂性。

而在以 Dynamo 为代表的非集中数据存储系统中，由于哈希方式的数据划分策略，使得系统中各个节点既作为存储节点也作为服务节点，服务迁移的过程伴随着大量的数据迁移，因而系统必须认真应对各种失效状态，以使失效恢复过程中尽量避免大规模存储数据迁移带来的系统开销。基于上述原因，在 Dynamo 中临时失效和永久失效被区别对待。其主要处理方式参见案例分析。

从上面两个方面的讨论，可以发现失效侦测技术的选择是一个与集群管理方式（集中式、非集中式）密切相关的问题；这种选择通常相对固定。而失效恢复策略的实现则是因应用而异。系统的设计者可以根据应用特性，权衡数据一致性、可用性以及系统性能等多方面因素选择较优的失效恢复策略。

2.3.5 海量数据存储的硬件支持

海量数据存储的实现离不开存储技术的不断发展，作为先进存储技术代表的 RAID、NAS、SAN 和 IP 技术均应用于海量数据的存储中。

1. 磁盘阵列（RAID）

RAID（Redundant Array of Independent Disks）是冗余的独立磁盘阵列的英文缩写。1988 年由美国加州大学 Berkeley 分校的 David Patterson 等人提出。冗余是为了补救错失、保证可靠性而采取的一种方法；独立是指阵列不在主机内而自成一个系统。一般将 RAID 分为不同级别，最常用的是 RAID0~PAID6。

（1）RAID0

RAID0 是最早出现的 RAID 模式，即 Data Stripping 数据分条技术，如图 2.1 所示。RAID 0 是组建磁盘阵列中最简单的一种形式，只需要两块以上的硬盘即可，成本低，可以提高整个磁盘的性能和吞吐量。RAID0 没有提供冗余或错误修复能力，但实现成本是最低的。

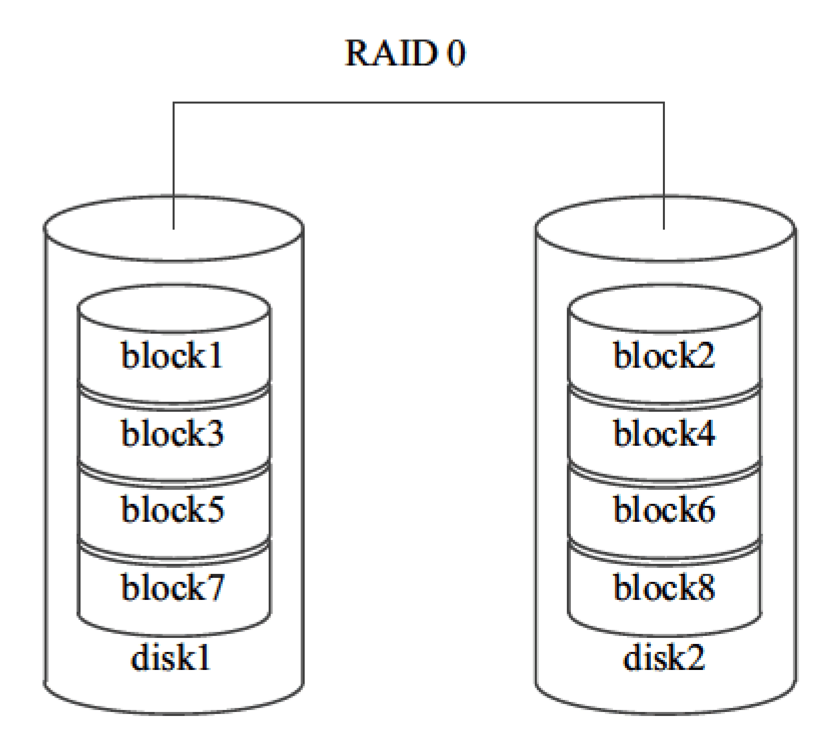


图 2.1　RAID0 组织结构图

RAID0 最简单的实现方式就是把 N 块同样的硬盘用硬件的形式通过智能磁盘控制器或用操作系统中的磁盘驱动程序以软件的方式串联在一起创建一个大的卷集。在使用中电脑数据依次写入到各块硬盘中，它的最大优点就是可以整倍地提高硬盘的容量。如使用了三块 80GB 的硬盘组建成 RAID0 模式，那么磁盘容量就会是 240GB。其速度方面，各个硬盘的速度完全相同。最大的缺点在于任何一块硬盘出现故障，整个系统将会受到破坏，可靠性仅为单独一块硬盘的 1/N。

虽然 RAID0 可以提供更多的空间和更好的性能，但是整个系统是非常不可靠的，如果出现故障，无法进行任何补救。所以，RAID0 一般只是在那些对数据安全性要求不高的情况下才被人们使用。

（2）RAID1

RAID1 称为磁盘镜像，如图 2.2 所示，原理是把一个磁盘的数据镜像到另一个磁盘上，也就是说数据在写入一块磁盘的同时，会在另一块闲置的磁盘上生成镜像文件，在不影响性能的情况下最大限度地保证系统的可靠性和可修复性，只要系统中任何一对镜像盘中至少有一块磁盘可以使用，甚至可以在一半数量的硬盘出现问题时系统都可以正常运行，当一块硬盘失效时，系统会忽略该硬盘，转而使用剩余的镜像盘读写数据，具备很好的磁盘冗余能力。虽然这样对数据来讲绝对安全，但是成本也会明显增加，磁盘利用率为 50%，以 4 块 80GB 容量的硬盘来讲，可利用的磁盘空间仅为 160GB。另外，出现硬盘故障的 RAID 系统不再可靠，应当及时更换损坏的硬盘，否则剩余的镜像盘也出现问题，整个系统就会崩溃。更换新盘后原有数据会需要很长时间同步镜像，外界对数据的访问不会受到影响，只是这时整个系统的性能有所下降。因此，RAID1 多用在保存关键性的重要数据的场合。

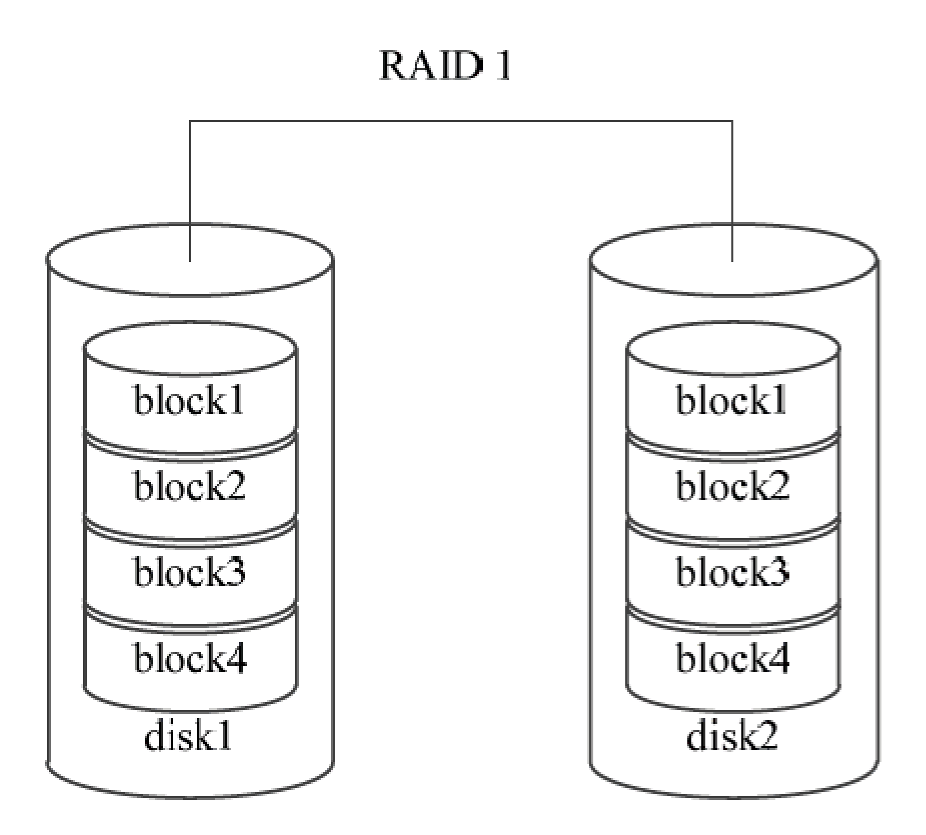


图 2.2　RAID1 组织结构图

RAID1 主要是通过二次读写实现磁盘镜像，所以磁盘控制器的负载也相当大，尤其是在需要频繁写入数据的环境中。为了避免出现性能瓶颈，使用多个磁盘控制器就显得很有必要。

（3）RAID2

从概念上讲，RAID2 同 RAID3 类似，两者都是将数据条块化分布于不同的硬盘上，条块单位为位或字节。然而 RAID2 使用一定的编码技术来提供错误检查及恢复。这种编码技术需要多个磁盘存放检查及恢复信息，使得 RAID2 技术实施更复杂。因此，在商业环境中很少使用。由于海明码的特点，它可以在数据发生错误的情况下将错误校正，以保证输出的正确。它的数据传送速率相当高，如果希望达到比较理想的速度，则最好提高保存校验码 ECC 码的硬盘，对于控制器的设计来说，它又比 RAID3、RAID4 或 RAID5 要简单。没有免费的午餐，这里也一样，要利用海明码，必须要付出数据冗余的代价。输出数据的速率与驱动器组中速度最慢的相等。

（4）RAID3：带奇偶校验码的并行传送

这种校验码与 RAID2 不同，只能查错不能纠错。它访问数据时一次处理一个带区，这样可以提高读取和写入速度。校验码在写入数据时产生并保存在另一个磁盘上。需要实现时用户必须要有三个以上的驱动器，写入速率与读出速率都很高，因为校验位比较少，因此计算时间相对而言比较少。用软件实现 RAID 控制将是十分困难的，控制器的实现也不是很容易。它主要用于图形（包括动画）等要求吞吐率比较高的场合。不同于 RAID 2，RAID 3 使用单块磁盘存放奇偶校验信息。如果一块磁盘失效，奇偶盘及其他数据盘可以重新产生数据。如果奇偶盘失效，则不影响数据使用。RAID 3 对于大量的连续数据可提供很好的传输率，但对于随机数据，奇偶盘会成为写操作的瓶颈。

（5）RAID4：带奇偶校验码的独立磁盘结构

RAID4 和 RAID3 很像，不同的是，它对数据的访问是按数据块进行的，也就是按磁盘进行的，每次是一个盘。在图上可以这么看，RAID3 是一次一横条，而 RAID4 是一次一竖条。它的特点和 RAID3 也挺像，不过在失败恢复时，它的难度可要比 RAID3 大得多了，控制器的设计难度也要大许多，而且访问数据的效率不怎么好。

RAID 技术主要包含 RAID0～RAID50 等数个规范，它们的侧重点各不相同。这里不再一一介绍。RAID 通过在多个磁盘上同时存储和读取数据来大幅提高存储系统的数据吞吐量

（Throughput）。在 RAID 中，可以让很多磁盘驱动器同时传输数据，而这些磁盘驱动器在逻辑上又是一个磁盘驱动器，所以使用 RAID 可以达到单个磁盘驱动器几倍、几十倍甚至上百倍的速率。这也是 RAID 最初想要解决的问题。因为当时 CPU 的速度增长很快，而磁盘驱动器的数据传输速率无法大幅提高，所以需要有一种方案解决二者之间的矛盾。RAID 最后成功了。

RAID 技术通过数据校验提供容错功能。普通磁盘驱动器无法提供容错功能，如果不包括写在磁盘上的 CRC（循环冗余校验）码的话。RAID 容错是建立在每个磁盘驱动器的硬件容错功能之上的，所以它提供更高的安全性。在很多 RAID 模式中都有较为完备的相互校验/恢复的措施，甚至是直接相互的镜像备份，从而大大提高了 RAID 系统的容错度，提高了系统的稳定冗余性。

2. 存储区域网（SAN）

存储区域网络（SAN）是通过专用高速网将一个或多个网络存储设备和服务器连接起来的专用存储系统，未来的信息存储将以 SAN 存储方式为主。SAN 在最基本的层次上定义为互连存储设备和服务器的专用光纤通道网络，它在这些设备之间提供端到端的通信，并允许多台服务器独立地访问同一个存储设备。与局域网（LAN）非常类似，SAN 提高了计算机存储资源的可扩展性和可靠性，使实施的成本更低、管理更轻松。与存储子系统直接连接服务器（称为直连存储或 DAS）不同，专用存储网络介于服务器与存储子系统之间，如图 2.3 所示。

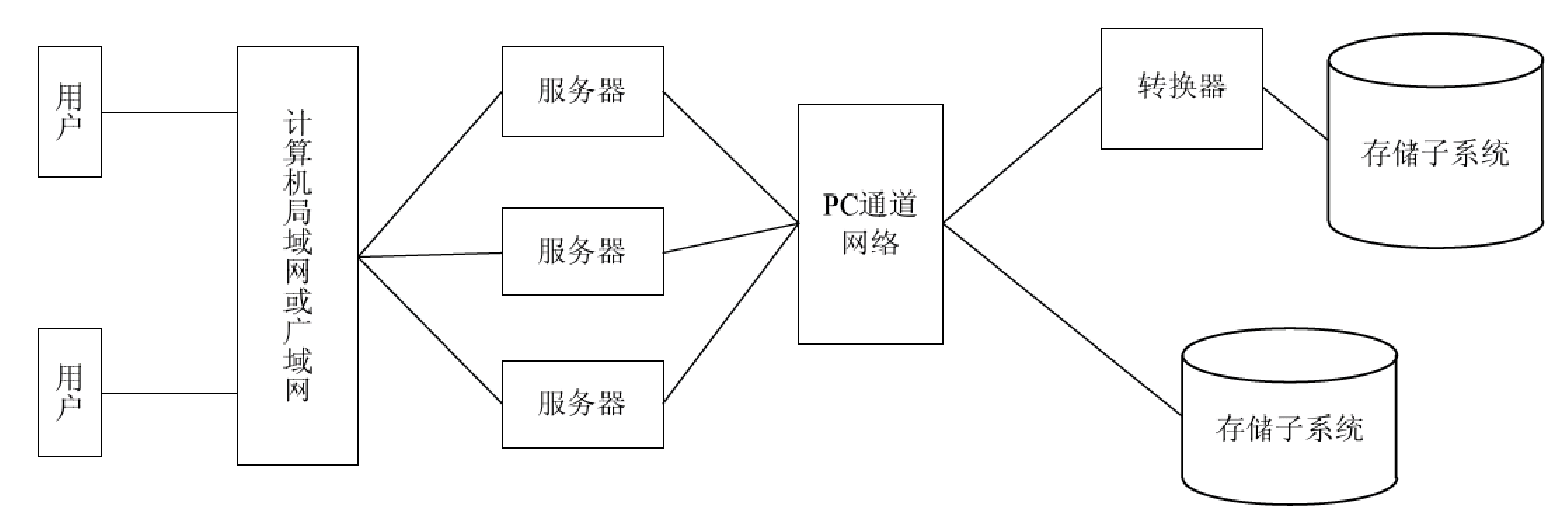


图 2.3　SAN 存储架构图

SAN 通过一个单独的网络把存储设备和服务器群相连。将数据存储管理集中在相对独立的存储区域网内，并提供内部任意节点之间的多路可选择数据交换。当有海量数据的存取需求时，数据通过 SAN 在相关服务器和后台存储设备之间高速传输。SAN 可在多种存储部件之间以及存储部件与交换机之间进行通信，将网络和设备的通信协议与传输介质隔离开，使系统在构建成本和复杂程度上大大降低，提高了网络利用率。另外，SAN 中容量扩展、数据迁移、远程容灾数据备份都比较方便。SAN 技术的存储设备性能高，提高了数据的可靠性和安全性，但设备的互操作性较差，构建、管理和维护成本高，只能提供存储空间共享而不能提供异构环境下的文件共享。

3. 网络附加存储（NAS）

NAS（Network Attached Storage，网络附加存储）是一种将分布、独立的数据整合为大型、

集中化管理的数据中心，以便于对不同主机和应用服务器进行访问的技术。

从结构上讲，NAS 是功能单一的精简型电脑，因此在架构上不像个人电脑那么复杂，在外观上就像家电产品，只需电源与简单的控制钮，如图 2.4 所示。

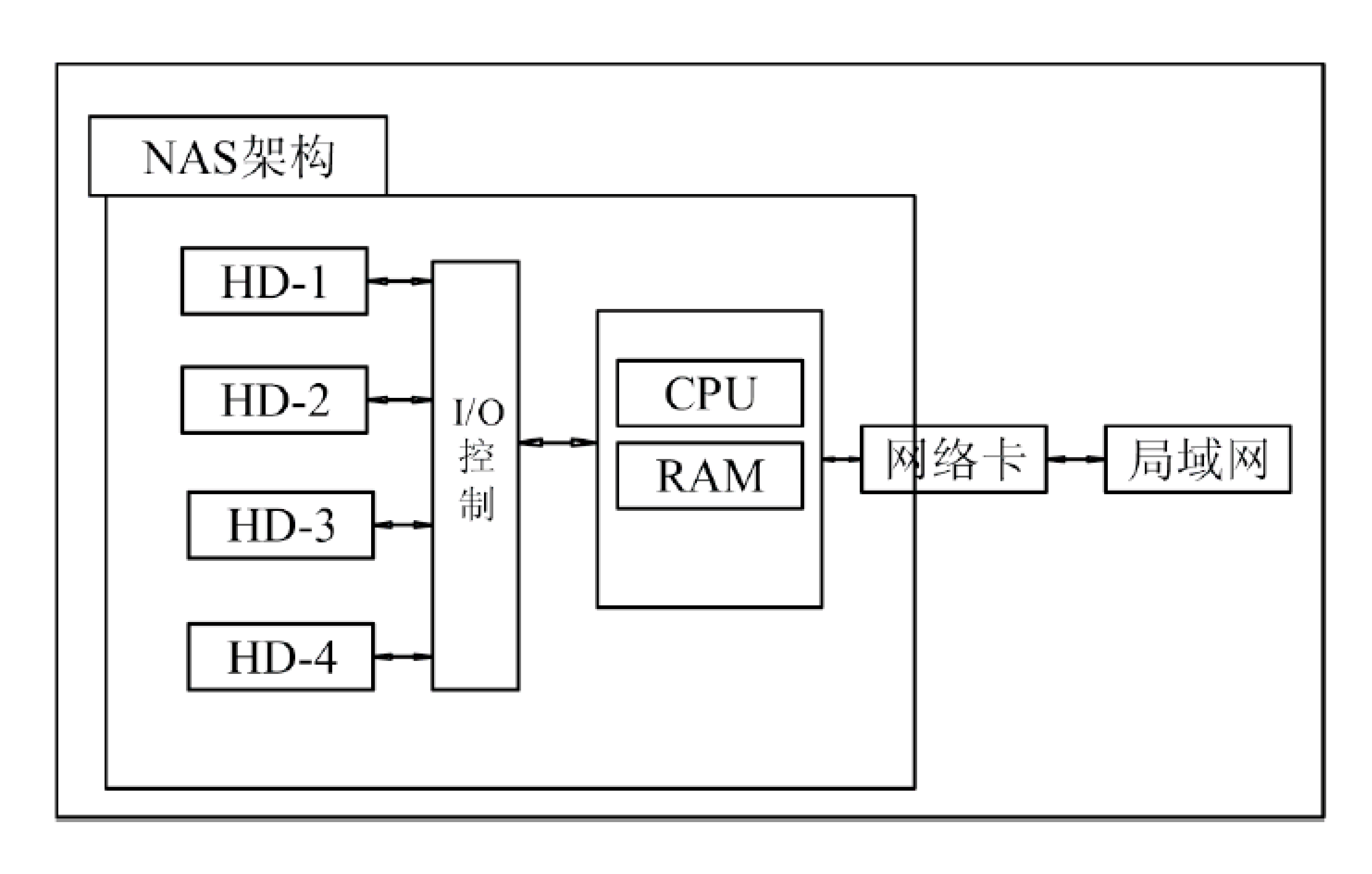


图 2.4　NAS 结构图

在以往，NAS 被视作网络上一种附加的存储设备，NAS 发展到现在，已经不单单是作为一种存储设备使用，更多的是作为数据备份和恢复的设备。这是由于 NAS 可以实现数据自动备份和恢复，而且由于 NAS 所使用的是嵌入式操作系统，有着很高的安全性，所以它的强项并不是所谓文件共享！真正适合文件共享的是 SAN 而不是 NAS，因为 SAN 是基于块（Block）级数据传输为目的的设备，它的性能比 NAS 高 N 倍，但价格也高了 N 倍，当然，用户同样可以使用 NAS 作为存储设备。NAS 真正面向的用户群是那些对历史数据极度依赖，一旦丢失就会带来灾难性后果的用户，如财务数据、资产数据、客户资料等，特别是 IT 资产投入预算相对较少的中小企业和用户。

NAS 备份存在以下优势：

（1）工业级标准的部件及制作工艺，让它们与其他存储设备区别开，这些在某种程度上可以体现在使用当中的稳定性、可靠性和安全性等方面；

（2）作为以数据存储、备份为专业的存储设备，采取精准的磁轨备份技术结合增量化、差异化体现了它们数据备份的有效性，数据还原的可靠性，部分网络附加存储（NAS）产品内嵌的高度智能化、自动化的专业数据备份软件高效地减少了数据备份对人的依赖；

（3）精简化的操作平台、IE 式的访问、操作模式在有效防范病毒、黑客的同时，大大提升了产品的易用性；

（4）账号式的管理模式简化了企业或个人对数据管理工作，企业通过账号式的管理可以根据企业内部不同角色的人员设置不同的数据存储、备份区域及操作权限，有效防患内部数据的混乱、泄密等状况的发生；

（5）磁盘阵列柜的特性在这些设备中得到了传承，通过组 RAID 的形式结合热插拔更换磁盘的特性，同时能够自动恢复设备内部数据使它的维护成本大大降低；

（6）没有地域限制、支持远程实时访问、备份、操作等特性，让它的部署更加容易，同时这

些特性在构建数据容灾系统中能够得到充分的体现；

（7）良好的异构平台兼容性，对各种操作平台如 Windows、Linnux、UNIX、Mac 等各平台间良好的兼容性，让它们可以轻松实现对各种基于上述操作平台的设备提供一机同时为多种平台进行数据备份的功能，在确保各项 IT 设备投入的有效性的同时避免了企业对 IT 设备重复投资造成资源的浪费的问题出现。

从本质上讲，NAS 是存储备份设备，不是服务器。NAS 不是简装版的文件服务器，它具有某些服务器没有的功能特性，例如增量备份、差异备份、磁轨备份等技术。服务器的作用是进行业务处理，存储设备（NAS）的作用是进行数据存储和备份，在一个完整的应用环境中应将两种设备有机地结合起来使用，服务器重点在前端，存储备份设备（NAS）的重点在后端。

4. IP 存储

随着网络存储技术的飞速发展，各种存储设备和技术正趋于融合。总有一天，现在的光纤和 SCSI 磁盘阵列、NAS 文件服务器、磁带库等设备都可以运行在一个统一标准的架构中。

IP 存储（Storage over IP，SoIP）在 IP 网络中传输块级数据，使得服务器可以通过 IP 网络连接 SCSI 设备，并且像使用本地的设备一样，无需关心设备的地址或位置。而网络连接则是以 IP 和以太网为骨干，这令人联想起今天耳熟能详的存储域网（SAN）结构。只是以廉价而成熟的 IP 和以太网技术，替代了光纤通道技术。

由于既有的成熟性和开放性，IP 存储技术，使企业在制定和实现“安全数据存储”的策略和方案时，有了更多的选择空间。例如远程的数据备份、数据镜像和服务器集群等领域，IP 存储的介入都可以大大丰富其内容。同时，IP 存储也消除了企业 IT 部门在设计传统 SAN 方案时，必须面对的产品兼容性和连接性方面的问题。最重要的是，基于 IP 存储技术的新型 SAN，兼具了传统 SAN 的高性能和传统 NAS 的数据共享优势，为新的数据应用方式提供了更加先进的结构平台。

在过去的一年中，存储和网络厂商的注意力主要集中在 IP 存储技术的两个方面，即存储隧道（Storage tunneling）和本地 IP 存储（Native IP-based Storage）。下面是这两个方面的一些粗略概况。顾名思义，存储隧道技术是将 IP 协议作为连接两个异地光纤 SAN 的隧道，用以解决两个 SAN 环境的互联问题。光纤通道协议帧被包裹在 IP 数据包中传输。数据包被传输到远端 SAN 后，由专用设备解包，还原成光纤通道协议帧。

由于这种技术提供的是两个 SAN 之间点到点的连接通信，从功能上讲，这是一种类似于光纤的专用连接技术。因此，这种技术也被称为黑光纤连接（Dark fiber optic links）。由于其专用性，使得这种技术实现起来成本较高，缺乏通用性，而且较大的延迟也对性能造成一定影响。其最大的优势在于，可以利用现有的城域网和广域网。这一优势，正好为炒作得沸沸扬扬，但至今无法充分利用的宽带资源，提供用武之地。 另一方面，虽然 IP 网络技术非常普及，其管理和控制机制也相对完善，但是，利用 IP 网络传输的存储隧道技术，却无法充分利用这些优势。其原因主要在于，嵌入 IP 数据包中的光纤通道协议帧。IP 网络智能管理工具不能识别这些数据，这使得一些很好的管理控制机制无法应用于这种技术，如目录服务、流量监控、QoS 等。因此，企业 IT 部门的系统维护人员，几乎不可能对包含存储隧道的网络环境，进行单一界面的统一集中化管理。

目前的存储隧道产品还有待完善，与光纤通道 SAN 相比，只能提供很小的数据传输带宽。例如，一个在光纤 SAN 上，用两到三个小时可以完成的传输过程，在两个光纤 SAN 之间以 OC-3 标准传输大约需要 14 个小时。这是目前存储隧道产品比较典型的传输速度。当然，这样的性能表现，不会限制到该技术在一些非同步功能中的应用。如远程的数据备份，就不一定需要很高的数据传输带宽。

总之，存储隧道技术，借用了一些 IP 网络的成熟性优势，但是并没有摆脱复杂而昂贵的光纤通道产品。

本地 IP 存储技术是将现有的存储协议，例如 SCSI 和光纤通道，直接集成在 IP 协议中，以使存储和网络可以无缝地融合。当然，这并不是指，可以在企业 IT 系统中，把存储网络和传统的 LAN，物理上合并成一个网络。而是指在传统的 SAN 结构中，以 IP 协议替代光纤通道协议，来构建结构上与 LAN 隔离，而技术上与 LAN 一致的新型 SAN 系统 IP-SAN。

这种 IP-SAN 中，用户可以在保证性能的同时，有效地降低成本，而且以往用户在 IP-LAN 上获得的维护经验、技巧都可以直接应用在 IP-SAN 上。俯拾皆是的 IP 网络工具，使 IP-SAN 的网络维护轻松而方便。同样，维护人员的培训工作，也不会像光纤技术培训那样庞杂而冗长。

设想一下，一个大型企业的 IT 部门引入了一项新技术，并以此构建了底层的大型存储系统。却不需要调整现有的网络和主机，不需要改变应用软件，不需要增加管理工具，甚至不需要过多的技术培训。现有的网络管理工具和人员，完全可以应付这一切。这是一个多么诱人的系统升级方案。

与存储隧道技术相比，本地 IP 存储技术具有显著的优势。首先，一体化的管理界面，使得 IP-SAN 可以和 IP 网络完全整合。其次，用户在这一技术中，面对的是非常熟悉的技术内容：IP 协议和以太网。而且，各种 IP 通用设备，保证了用户可以具有非常广泛的选择空间。事实上，由于本地 IP 存储技术的设计目标，就是充分利用现有设备，传统的 SCSI 存储设备和光纤存储设备，都可以在 IP-SAN 中利用起来。

本地 IP 存储技术，更进一步地模糊了本地存储和远程存储的界限。在 IP-SAN 中，只要主机和存储系统都能提供标准接口，任何位置的主机就都可以访问任何位置的数据，无论是在同一机房中，相隔几米，还是数公里外的异地。

访问的方式可以是类似 NAS 结构中，通过 NFS、CIFS 等共享协议访问，也可以是类似本地连接和传统 SAN 中，本地设备级访问。

2.4 数据存储技术的实现与工具

基于 2.2.2 节的介绍，在本小节，主要介绍一些 NoSQL 数据存储工具。

2.4.1 集中式数据存储管理系统 Bigtable

Bigtable 是 Google 设计的分布式数据存储系统，用来处理海量数据的一种非关系型数据库。

Bigtable 是非关系的数据库，是一个稀疏的、分布式的、持久化存储的多维度排序 Map。Bigtable 的设计目的是可靠地处理 PB 级别的数据，并且能够部署到上千台机器上。Bigtable 已经实现了下面的几个目标：适用性广泛、可扩展、高性能和高可用性。Bigtable 已经在超过 60 个 Google 的产品和项目上得到了应用，包括 Google Analytics、Google Finance、Orkut、Personalized Search、Writely 和 GoogleEarth。这些产品对 Bigtable 提出了迥异的需求，有的需要高吞吐量的批处理，有的则需要及时响应，快速返回数据给最终用户。它们使用的 Bigtable 集群的配置也有很大的差异，有的集群只有几台服务器，而有的则需要上千台服务器、存储几百 TB 的数据。

在很多方面，Bigtable 和数据库很类似：它使用了很多数据库的实现策略。并行数据库和内存数据库已经具备可扩展性和高性能，但是 Bigtable 提供了一个和这些系统完全不同的接口。Bigtable 不支持完整的关系数据模型；与之相反，Bigtable 为客户提供了简单的数据模型，利用这个模型，客户可以动态控制数据的分布和格式，用户也可以自己推测底层存储数据的位置相关性。数据的下标是行和列的名字，名字可以是任意的字符串。Bigtable 将存储的数据都视为字符串，但是 Bigtable 本身不去解析这些字符串，客户程序通常会把各种结构化或者半结构化的数据串行化到这些字符串里。通过仔细选择数据的模式，客户可以控制数据的位置相关性。最后，可以通过 BigTable 的模式参数来控制数据是存放在内存中、还是硬盘上。

Bigtable 是设计用来管理那些可能达到大小很大（比如可能是存储在数千台服务器上的数 PB 数据）的结构化数据的分布式存储系统。Google 的很多项目都将数据存储在 Bigtable 中，比如网页索引、Google 地球、Google 金融。这些应用对 Bigtable 提出了很多不同的要求，无论是数据大小（从单纯的 URL 到包含图片附件的网页）还是延时需求。尽管存在这些各种不同的需求，Bigtable 成功地为 Google 的所有这些产品提供了一个灵活的、高性能的解决方案。

1. BigTable 数据模型

Bigtable 是一个稀疏的、分布式的一致性多维有序 map。这个 map 是通过行关键字、列关键字以及时间戳进行索引的；map 中的每个值都是一个未经解释的字节数组。

```
(row:string,column:string,time:int64) -> string
```

如果想保存一份可以被很多工程使用的一大集网页及其相关信息的拷贝。我们把这个表称为 webtable，在这个表中，可以使用 URL 作为行关键字，网页的各种信息作为列名称，将网页的内容作为表的内容存储：获取的时候还需要在列上加上时间戳，如图 2.5 所示。

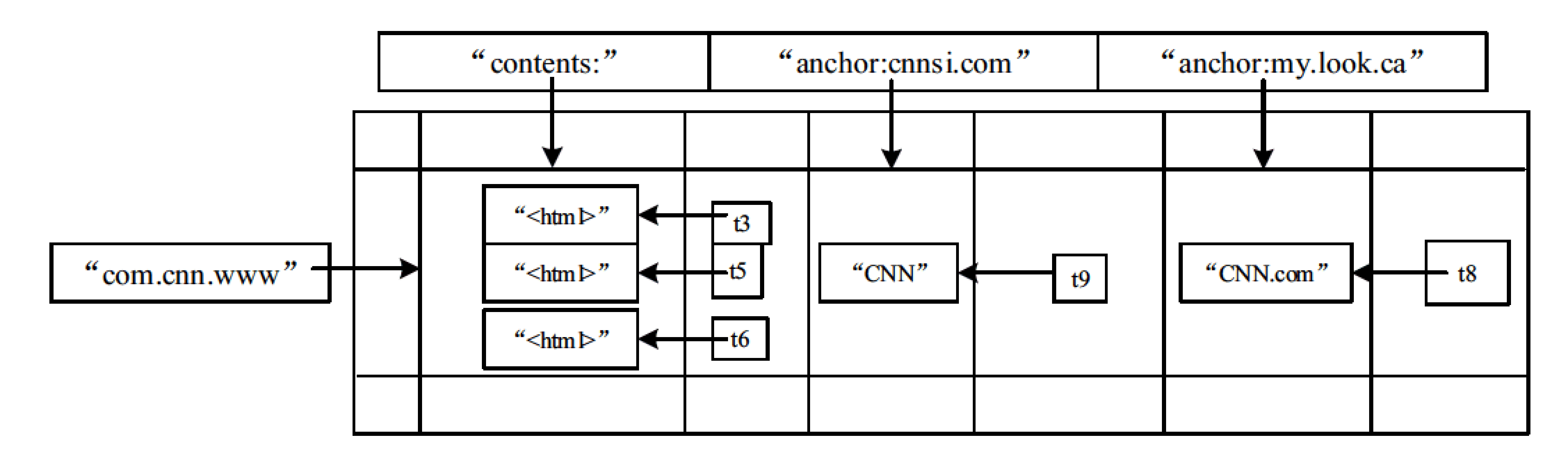


图 2.5　Bigtable 存储结构图

表中的行关键字是大字符串（目前最大可以到 64KB，尽管对于大多数用户来说最常用的是10~100B）。在一个行关键字下的数据读写是原子性的（无论这一行有多少个不同的列被读写），这个设计使得用户在对相同行的并发更新出现时，更容易理解系统的行为。

Bigtable 按照行关键字的字典序来维护数据。行组是可以动态划分的。每个行组叫做一个tablet，是数据存放以及负载平衡的单位。这样，对于一个短的行组的读就会很有效，而且只需要与少数的机器进行通信。客户端可以通过选择行关键字来利用这个属性，这样它们可以为数据访问得到好的 locality。比如，在 webtable 里，相同域名的网页可以通过将 URL 中的域名反转而使它们放在连续的行里来组织到一块。比如将网页 maps.Google.com/index.html 的数据存放在关键字 com.Google.maps/index.html 下。将相同域名的网页存储在邻近位置可以使对主机或域名的分析更加有效。

不同的列关键字可以被分组到一个集合，把这样的一个集合称为一个列族，它是基本的访问控制单元。存储在同一个列族的数据通常是相同类型的。在数据能够存储到某个列族的列关键字下之前，首先必须要创建该列族。假设在一个表中不同列族的数目应该比较小（最多数百个），而且在操作过程中这些列族应该很少变化。与之相比，一个表的列数目可以没有限制。

一个列关键字是使用如下的字符来命名的：family:qualifier。列族名称必须是可打印的，但是 qualifier 可能是任意字符串。比如 webtable 有一个列族是 language，它存储了网页所使用的语言。在 language 列族里，只使用了一个列关键字，里面存储了每个网页的 language id。该表的另一个列族是 anchor，在该列族的每个列关键字代表一个单独的 anchor，如图 2.5 所示。Qualifier 是站点的名称，里面的内容是链接文本。

访问控制以及磁盘和内存分配都是在列族级别进行的。在 webtable 这个例子中，这些控制允许管理不同类型的应用：一些可能会添加新的基础数据，一些可能读取这些基础数据来创建新的列族，一些可能只需要查看现有数据（甚至可能因为隐私策略而无法查看所有现有数据）。

Bigtable 里的每个 cell 可以包含相同数据的多个版本；这些不同的版本是通过时间戳索引的。Bigtable 的时间戳是一个 64 位的整数。它们可以由 Bigtable 来赋值，在这种情况下它们以毫秒来代表时间。也可以由客户端应用程序显式分配。应用程序为了避免冲突必须能够自己生成唯一的时间戳。一个 cell 的不同版本是按照时间戳降序排列，这样最近的版本可以被首先读到。

为了使不同版本的数据管理更简单，支持通过两种方法针对每个列族进行设定，来告诉 Bigtable 如何对 cell 中的数据版本进行自动的垃圾回收：用户可以指定最近的哪几个版本需要保存，或者保存那些足够新的版本（比如只保存那些最近 7 天写的数据）。

在 webtable 中，将被抓取网页的时间戳存储在 contents 里：这些时间说明了这些网页的不同版本分别是在何时被抓取的。前面描述的垃圾回收机制，可以只保存每个网页最近的三个版本。

2. BigTable 的基础构件

BigTable 建立在 Google 的其他几个设施之上。Bigtable 使用 GFS 来存储日志和数据文件。Bigtable 集群通常运行在一个运行着大量其他分布式应用的共享机器池上。Bigtable 依赖于一个集群管理系统进行 job 调度和共享机器上的资源管理，处理机器失败以及监控机器状态。

Bigtable 内部采用 Google SSTable 文件格式来存储数据。一个 SSTable 提供了一个一致性的、

有序的从 key 到 value 的不可变 map，key 和 value 都是任意的字节串。操作通常是通过一个给定的 key 来查找相应的 value，或者在一个给定的 key range 上迭代所有的 key/value 对。每个 SSTable 内部包含一系列的块（通常每个块大小是 64KB，该大小是可配置的）。一个块索引是用来定位 block 的，当 SSTable 打开时该索引会被加载到内存。一次查找可以通过一次磁盘访问完成：首先通过在内存中的索引进行一次二分查找找到相应的块，然后从磁盘中读取该块。另外，一个 SSTable 可以被完全映射到内存，这样不需要接触磁盘就可以执行所有的查找和扫描。

Bigtable 依赖于一个高可用的一致性分布式锁服务 Chubby。Chubby 由 5 个活动副本组成，其中的一个选为 master 处理请求。当超过半数的副本运行并且可以相互通信时，该服务就是可用的。Chubby 使用 Paxos 算法来保证在出现失败时，副本的一致性。Chubby 提供了一个由目录和小文件组成的名字空间。每个文件或者目录可以当作一个锁来使用，对于一个文件的读写是原子性的。Chubby 的客户端库为 Chubby 文件提供一致性缓存。每个 Chubby 客户端维护着一个与 Chubby 服务的会话。如果在租约有效时间内无法更新会话的租约，客户端的会话就会过期。当一个客户端会话过期后，它会丢失所有的锁和打开的文件句柄。Chubby 客户端也会在 Chubby 文件和目录上注册回调函数来处理这些变更或者会话的过期。

3. Table 放置

BigTable 使用一个类似于 B+树的三级结构来存储 tablet 的放置信息，如图 2.6 所示。

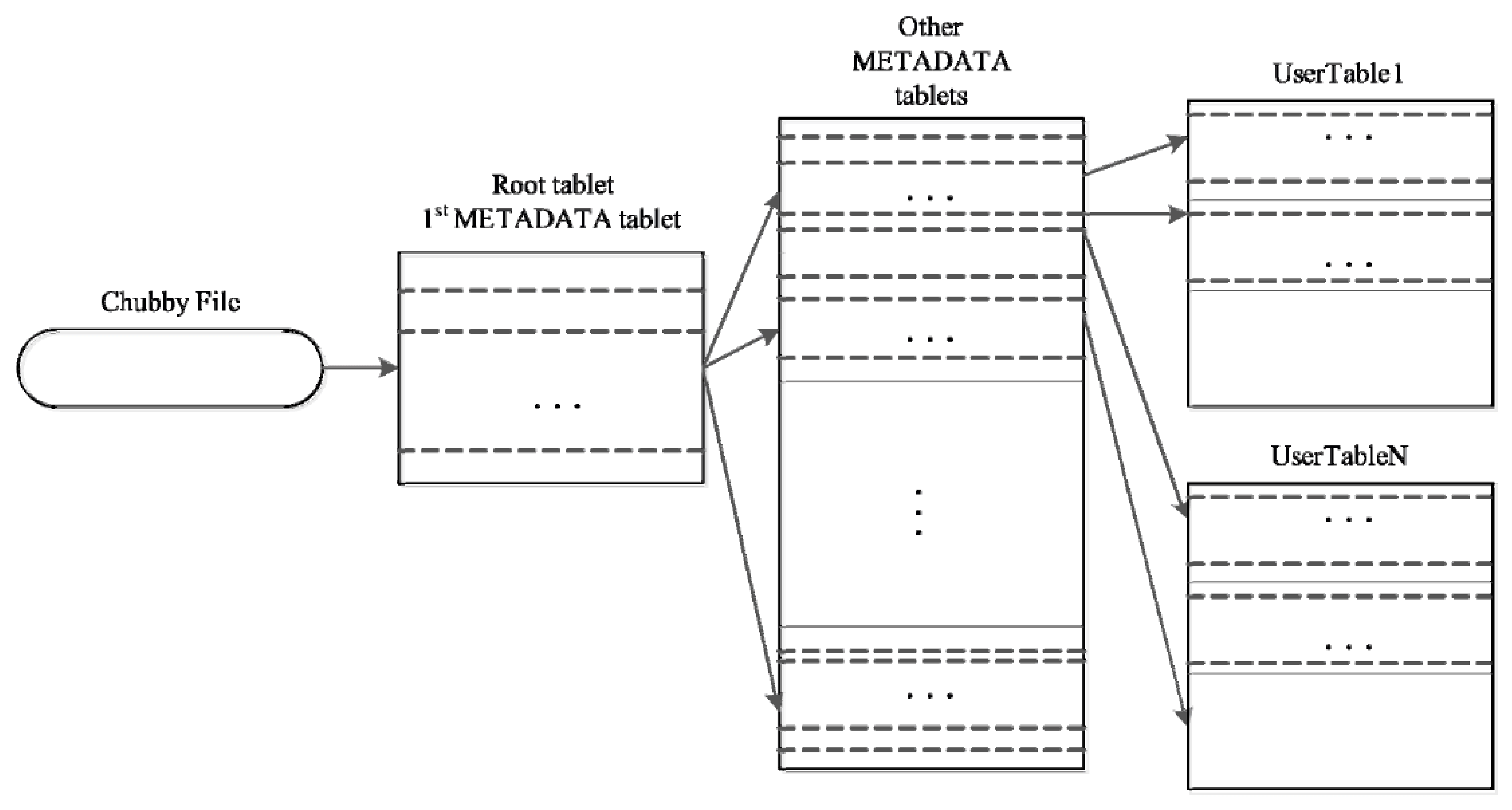


图 2.6　METADATA 组织结构图

首先是第一层，Chubby file。这一层是一个 Chubby 文件，它保存着 root tablet 的位置。这个 Chubby 文件属于 Chubby 服务的一部分，一旦 Chubby 不可用，就意味着丢失了 root tablet 的位置，整个 Bigtable 也就不可用了。

第二层是 root tablet。root tablet 其实是元数据表（METADATA tablet）的第一个分片，它保存着元数据表其他片的位置。root tablet 很特别，为了保证树的深度不变，root tablet 从不分裂。

第三层是其他的元数据片，它们和 root tablet 一起组成完整的元数据表。每个元数据片都包含了许多用户片的位置信息。

可以看出整个定位系统其实只是两部分，一个 Chubby 文件，一个元数据表。注意元数据表虽然特殊，但也仍然服从前文的数据模型，每个分片也都是由专门的片服务器负责，这就是不需要主服务器提供位置信息的原因。客户端会缓存片的位置信息，如果在缓存里找不到一个片的位置信息，就需要查找这个三层结构了，包括访问一次 Chubby 服务，访问两次片服务器。

root tablet 大小为 128M，每个行 1KB，那么它就可以存储 128*2^20/2^10=128*2^10 个 METADATA tablet，同样的，每个 METADATA tablet 可以存储 128*2^10 个普通 tablet，这样总共可以存储 128*2^10*128*2^10 即 2^34 个普通 tablet，每个 tablet 有将近 1KB 元数据，这样算起来存储这些元信息就需要 4TB 的数据，所以该 METADATA 表是分布式地放在不同的机器上的，同时也不可能全部放入内存，而是采用与普通的表一样的存储方式，放在 GFS 上。

4. Table 分配

在任何一个时刻，一个 Tablet 只能分配给一个 Tablet 服务器。Master 服务器记录了当前有哪些活跃的 Tablet 服务器、哪些 Tablet 分配给了哪些 Tablet 服务器、哪些 Tablet 还没有被分配。当一个 Tablet 还没有被分配、并且刚好有一个 Tablet 服务器有足够的空闲空间装载该 Tablet 时，Master 服务器会给这个 Tablet 服务器发送一个装载请求，把 Tablet 分配给这个服务器。

BigTable 使用 Chubby 跟踪记录 Tablet 服务器的状态。当一个 Tablet 服务器启动时，它在 Chubby 的一个指定目录下建立一个有唯一性名字的文件，并且获取该文件的独占锁。Master 服务器实时监控着这个目录（服务器目录），因此 Master 服务器能够知道有新的 Tablet 服务器加入了。如果 Tablet 服务器丢失了 Chubby 上的独占锁——比如由于网络断开导致 Tablet 服务器和 Chubby 的会话丢失——它就停止对 Tablet 提供服务。Chubby 提供了一种高效的机制，利用这种机制，Tablet 服务器能够在不增加网络负担的情况下知道它是否还持有锁。只要文件还存在，Tablet 服务器就会试图重新获得对该文件的独占锁；如果文件不存在了，那么 Tablet 服务器就不能再提供服务了，它会自行退出。当 Tablet 服务器终止时（比如，集群的管理系统将运行该 Tablet 服务器的主机从集群中移除），它会尝试释放它持有的文件锁，这样一来，Master 服务器就能尽快把 Tablet 分配到其他的 Tablet 服务器。

Master 服务器负责检查一个 Tablet 服务器是否已经不再为它的 Tablet 提供服务，并且要尽快重新分配它加载的 Tablet。 Master 服务器通过轮询 Tablet 服务器文件锁的状态来检测何时 Tablet 服务器不再为 Tablet 提供服务。如果一个 Tablet 服务器报告它丢失了文件锁，或者 Master 服务器最近几次尝试和它通信都没有得到响应，Master 服务器就会尝试获取该 Tablet 服务器文件的独占锁；如果 Master 服务器成功获取了独占锁，那么就说明 Chubby 是正常运行的，而 Tablet 服务器要么是宕机了，要么是不能和 Chubby 通信了，因此，Master 服务器就删除该 Tablet 服务器在 Chubby 上的服务器文件以确保它不再给 Tablet 提供服务。一旦 Tablet 服务器在 Chubby 上的服务器文件被删除了，Master 服务器就把之前分配给它的所有的 Tablet 放入未分配的 Tablet 集合中。为了确

保 Bigtable 集群在 Master 服务器和 Chubby 之间网络出现故障的时候仍然可以使用，Master 服务器在它的 Chubby 会话过期后主动退出。但是不管怎样，如同前面所描述的，Master 服务器的故障不会改变现有 Tablet 在 Tablet 服务器上的分配状态。

当集群管理系统启动了一个 Master 服务器之后，Master 服务器首先要了解当前 Tablet 的分配状态，之后才能够修改分配状态。 Master 服务器在启动的时候执行以下步骤:(1)Master 服务器从 Chubby 获取一个唯一的 Master 锁，用来阻止创建其他的 Master 服务器实例；(2)Master 服务器扫描 Chubby 的服务器文件锁存储目录，获取当前正在运行的服务器列表；(3)Master 服务器和所有的正在运行的 Tablet 表服务器通信，获取每个 Tablet 服务器上 Tablet 的分配信息；(4)Master 服务器扫描 METADATA 表获取所有的 Tablet 的集合。在扫描的过程中，当 Master 服务器发现了一个还没有分配的 Tablet，Master 服务器就将这个 Tablet 加入未分配的 Tablet 集合等待合适的时机分配。

可能会遇到一种复杂的情况：在 METADATA 表的 Tablet 还没有被分配之前是不能够扫描它的。因此，在开始扫描之前（步骤 4），如果在第三步的扫描过程中发现 Root Tablet 还没有分配，Master 服务器就把 Root Tablet 加入到未分配的 Tablet 集合。这个附加操作确保了 Root Tablet 会被分配。由于 Root Tablet 包括了所有 METADATA 的 Tablet 的名字，因此 Master 服务器扫描完 Root Tablet 以后，就得到了所有的 METADATA 表的 Tablet 的名字。

保存现有 Tablet 的集合只有在以下事件发生时才会改变：建立了一个新表或者删除了一个旧表、两个 Tablet 被合并了或者一个 Tablet 被分割成两个小的 Tablet。Master 服务器可以跟踪记录所有这些事件，因为除了最后一个事件外的两个事件都是由它启动的。Tablet 分割事件需要特殊处理，因为它由 Tablet 服务器启动。在分割操作完成之后，Tablet 服务器通过在 METADATA 表中记录新的 Tablet 的信息来提交这个操作；当分割操作提交之后，Tablet 服务器会通知 Master 服务器。如果分割操作已提交的信息没有通知到 Master 服务器（可能两个服务器中有一个宕机了），Master 服务器在要求 Tablet 服务器装载已经被分割的子表的时候会发现一个新的 Tablet。通过对比 METADATA 表中 Tablet 的信息，Tablet 服务器会发现 Master 服务器要求其装载的 Tablet 并不完整，因此，Tablet 服务器会重新向 Master 服务器发送通知信息。

5. Tablet 服务

如图 2.7 所示，Tablet 的持久化状态信息保存在 GFS 上。更新操作提交到 REDO 日志中。在这些更新操作中，最近提交的那些存放在一个排序的缓存中，我们称这个缓存为 memtable；较早的更新存放在一系列 SSTable 中。为了恢复一个 Tablet，Tablet 服务器首先从 METADATA 表中读取它的元数据。Tablet 的元数据包含了组成这个 Tablet 的 SSTable 的列表，以及一系列的 Redo Point，这些 Redo Point 指向可能含有该 Tablet 数据的已提交的日志记录。Tablet 服务器把 SSTable 的索引读进内存，之后通过重复 Redo Point 之后提交的更新来重建 memtable。

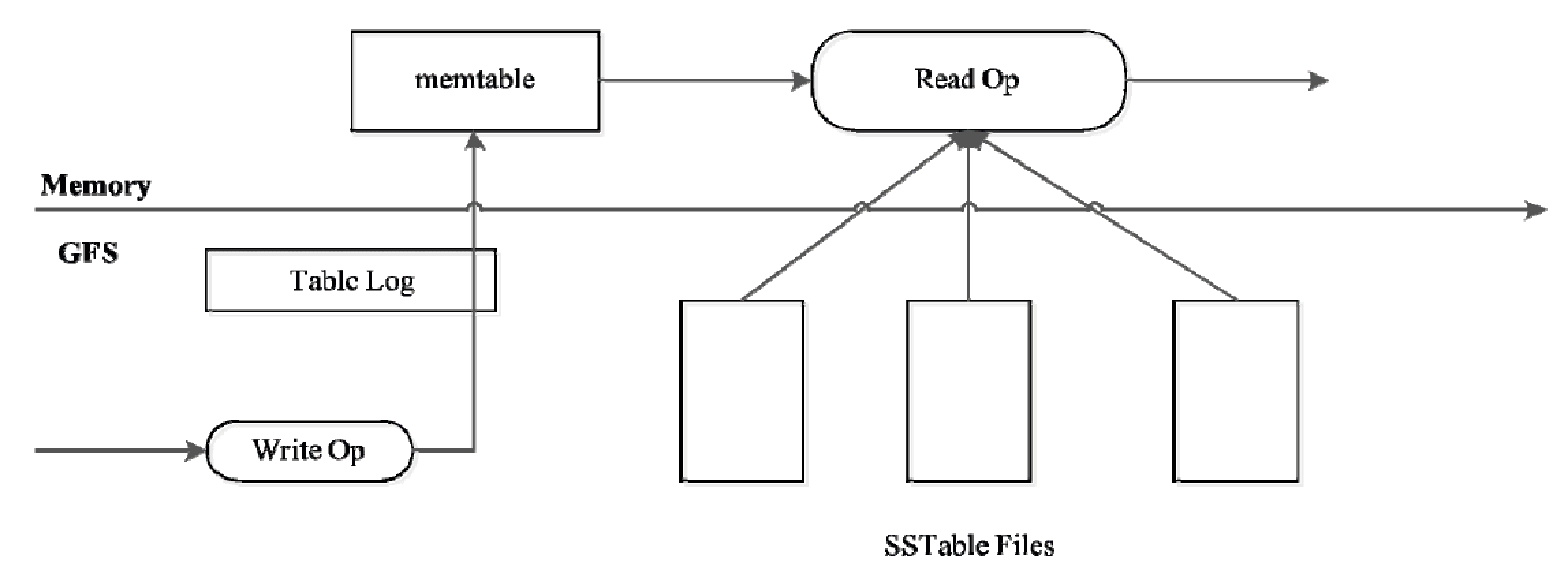


图 2.7 Table 的 GFS 存储

当对 Tablet 服务器进行写操作时，Tablet 服务器首先要检查这个操作格式是否正确、操作发起者是否有执行这个操作的权限。权限验证的方法是通过从一个 Chubby 文件里读取出来的具有写权限的操作者列表来进行验证（这个文件几乎一定会存放在 Chubby 客户缓存里）。成功的修改操作会记录在提交日志里。可以采用批量提交方式来提高包含大量小的修改操作的应用程序的吞吐量。当一个写操作提交后，写的内容插入到 memtable 里面。

当对 Tablet 服务器进行读操作时，Tablet 服务器会作类似的完整性和权限检查。一个有效的读操作在一个由一系列 SSTable 和 memtable 合并的视图里执行。由于 SSTable 和 memtable 是按字典排序的数据结构，因此可以高效地生成合并视图。

当进行 Tablet 的合并和分割时，正在进行的读写操作能够继续进行。

6. BigTable 优化策略

前面一节描述的实现需要大量的技巧来到达用户所需要的高性能、可用性、可靠性。这一节更细节地描述实现的各个部分，着重讲述使用的技术。

（1）Locality groups（对应一个 SSTable）

用户可以将多个列族组织为一个 Locality group。对于每个 tablet 里的每个 Locality group 都会生成一个单独的 SSTable。将那些经常不被访问的列族分离到一个独立的群组可以增加访问的效率。比如，webtable 的关于网页的元数据（比如语言，校验和）可放到一个 Locality group，网页内容可以放到另一个群组里，这样一个访问元数据的应用程序就不需要读取所有网页的内容。

另外，一些有用的 tuning 参数也可以以 Locality group 为单位进行设置。比如一个 Locality group 可以声明为放入内存的。对于声明为放入内存的 Locality group 的 SSTable 在需要时才会加载到 tablet 服务器的内存中。一旦加载之后，对于该 Locality group 的访问就不需要访问磁盘。这个特点对于那些需要经常访问的小片数据很有用：比如在 Bigtable 内部我们将它应用在 METADATA 表的 location 列族上。

（2） 压缩

用户可以控制对于一个 Locality group 的 SSTables 是否进行压缩，以及使用哪种压缩格式。

用户指定的压缩格式会应用在 SSTable 的每个块上（块大小可以通过一个 Locality group 的参数进行控制）。对于每个块单独进行压缩，尽管这使我们丢失了一些空间，但是我们不需要解压整个文件就可以读取 SSTable 的部分内容。很多用户使用一个两遍压缩模式，第一遍压缩使用 Bentley and McIlroy 模式，该模式在一个很大的窗口大小里压缩普通的长字符串。第二遍压缩使用了一个快速压缩算法，该算法在一个小的 16KB 窗口大小内查找重复。这两遍压缩都很快速，压缩速率在 100~200MB/s，解压速率在 400~1000MB/s。

尽管在选择压缩算法时，我们更重视速率而不是空间的减少，但是这个两遍压缩模式工作的出奇地好。比如，在 webtable 里使用这种压缩模式存储网页内容。实验中，在一个压缩的 Locality group 里存储了大量文档。为了实验目的，将文档的版本数限制为 1。该压缩模式达到了 10 :1 的压缩率。这比通常的 Gzip 对于 HTML 网页的 3:1 或 4:1 的压缩率要好多了。这是由我们的 webtable 的行分布方式造成的：来自相同主机的网页被存储在相邻的位置。这使 Bentley and McIlroy 算法可以识别出来自相同站点的大量固有模式。很多应用程序，不仅仅是 webtable，选择的行名称使得类似数据会聚集在一起，因此达到了很好的压缩率。当我们在 Bigtable 中存储相同值的多个版本时压缩率会更好。

7. BigTable 实际应用

（1）Google Analytics

Google Analytics 是一个帮助站长分析站点的流量模式的服务。它提供整体的统计，比如每天内的不同访问者数目，每个 URL 每天的访问数，以及一些站点反馈报告，比如那些打开了某特定页面后的访问者进行交易的比例。

为了使用这项服务，站长需要在他们的页面里嵌入一个小 JavaScript 程序。它会将关于请求的各项信息记录在 Google Analytics 里，比如用户标识以及该网页被获取的信息。Google Analytics 对这些数据进行分析并给站长使用。

我们简要描述 Google Analytics 使用的两个表。原始的点击表（大概 200TB）为每个终端用户会话维护一条记录。行的名称是一个由站点名称以及会话创建时间组成的元组。这种 schema 使得那些访问相同站点的会话是相邻的，而且是按照时间排序的，这个表格可以压缩到原始大小的 14%。

摘要表格（大概 20TB）包含对每个站点各种预定义的摘要。这个表格是通过周期性调度的 MapReduce jobs 从原始点击表生成出来的。每个 MapReduce job 从原始点击表抽取出最近的会话数据。系统整体的吞吐率由 GFS 的吞吐率决定，这个表格可以压缩为原始大小的 29%。

（2）Google 地球

Google 提供一组服务，使得用户既可以通过网页也可以通过 Google 地球客户端软件访问地球表面的高分辨率卫星图像。这些产品允许用户浏览地球表面：他们可以在不同的分辨率上拍摄、观看、注释卫星图像。系统使用一个表来预处理数据，用另外一些表来为用户提供数据。

预处理流程使用一个表来存储原始图像。在预处理期间，图像被整理，然后合并为最终的服务数据。这个表大概有 70TB 数据，因此需要通过硬盘提供服务。图像本来已进行了压缩，因此

Bigtable 的压缩选项被关掉了。

图像表里的每一行对应一个地理区域，行的命名方式保证相邻的地理位置会被存储到邻近的位置。表格包含一个列族来保存每个区域的数据源。这个列族有大量的列：每个列保存一个原始数据图片。因为每个区域仅仅从是几个图片构建出来的，因此这个列族是很稀疏的。

这个预处理流程依赖于 MapReduce 在 Bigtable 上进行数据转换。在这些 MapReduce jobs 运行期间，整个系统每个 tablet 服务器的数据处理速度超过 1MB/s。

服务系统使用一个表来索引存储在 GFS 中的数据。这个表相对小一些（大概 500GB），但是每个数据中心每秒必须要为数千个查询提供低延时服务。因此，这个表实际上会占用几百个 tablet 服务器，包含许多 in-memory 列族。

（3）个性化搜索

个性化搜索是一个用来记录用户在 Google 很多产品（比如网页搜索、图像新闻搜索）的查询和单击记录的可选服务。用户可以通过浏览搜索历史来查看过去的查询和单击，可以通过 Google 的历史记录得到个性化的搜索结果。个性化搜索将每个用户的数据保存在 Bigtable 里。每个用户具有唯一的用户 id，并分配给他一个以用户 id 命名的行。所有的用户动作保存在表中。每个类型的动作用一个独立的列族保存（比如有一个列族保存所有的网页查询）。每个数据元素使用对应动作的发生时间作为它在 Bigtable 中的时间戳。个性化搜索通过在 Bigtable 上的 MapReduce 产生用户特征。这些用户特征被用来个性化实时搜索结果。

为了提高可用性和降低用户延时，个性化搜索数据备份在多个 Bigtable 集群上。个性化搜索团队起初自己在 Bigtable 之上建立了一个用户端备份机制来保证所有备份的一致性。现在的系统已经使用一个内建到服务端的备份子系统了。

个性化搜索存储系统的设计允许其他团队在他们的列中添加新的用户信息。现在这个系统已经被很多其他需要存储用户配置和设置信息的 Google 产品使用。多个团队间共享一个表使得表具有大量的列族。为了支持共享，给 Bigtable 添加了一个简单的 quota 机制来限制共享表的用户的存储消耗。这个机制为不同产品团队使用该系统进行用户信息存储提供了隔离性。

2.4.2 非集中式的大规模数据管理系统 Dynamo

Dynamo 是一个基于分布式哈希的非集中式的大规模数据管理系统。在 Dynamo 中，数据按照键/值对组织，主要面向原始数据的存储。这种架构下，系统中每个节点都能相互感知，自我管理性能较强，没有单点失效。

Dynamo 有如下特点：

- 简单的存取模式，只支持 KV 模式的数据存取，同时特定于小于 1M 的数据。
- 高可用性，即便在集群中部分机器故障，网络中断、甚至是整个机房下线，仍能保证用户对数据的读写。
- 高可扩展性，除了能够跨机房部署外，可动态增加、删除集群节点，同时对正常集群影响很小。

- 数据的高可用性大于数据的一致性，短时间的数据不一致是可以容忍的，采用最终一致性来保证数据的高可用。
- 服务于内网，数据间没有隔离。
- 服务保证条约，在 Amazon 一切皆服务的原则下，每个模块都要为其使用者提供服务时间保证，比如在每秒 500 个请求的压力下，99.9%的请求要在 300ms 内返回。

接下来详细介绍为了完成上述目标，Dynamo 中所采用的技术方案。

1. 数据划分(Partition)

Dynamo 作为一个分布式的存储，必须能够将用户输入的数据划分到集群中的不同机器中存储。传统的划分方式都是类 Hash 的，即按照数据中的某些特征值做 Hash（一般都是取模），使数据映射到下边的机器中。但是这种方式有一个很大的缺点，那就是当集群中机器增加的时候，之前的 Hash 全部都失效了，再 Hash 的成本很大，也就制约了集群的扩展。

Dynamo 中采用了“一致性 Hash”来解决这个问题。一致性哈希算法将哈希函数的值域首尾相接形成一个环。系统中的每一个节点在值域区间内随机分配一个值以作为节点在环中的位置。这些值将环划分成不同的范围，分配给集群系统中的不同节点进行管理。当对数据进行请求（读取/插入）时，通过计算该键/值对中键的哈希值，沿环顺时针找到第一个节点进行服务请求。

如图 2.8 所示。4 个节点被随机分布到环上，当数据记录的哈希键（key）落入区间（节点 3，节点 1）时沿环顺时针找到节点 1 进行存储。依次类推，节点 2 存储哈希值落入区间（节点 1，节点 2）的数据，节点 4 存储哈希值落入区间（节点 2，节点 4）的数据，等等。其优势主要体现在以下两个方面：

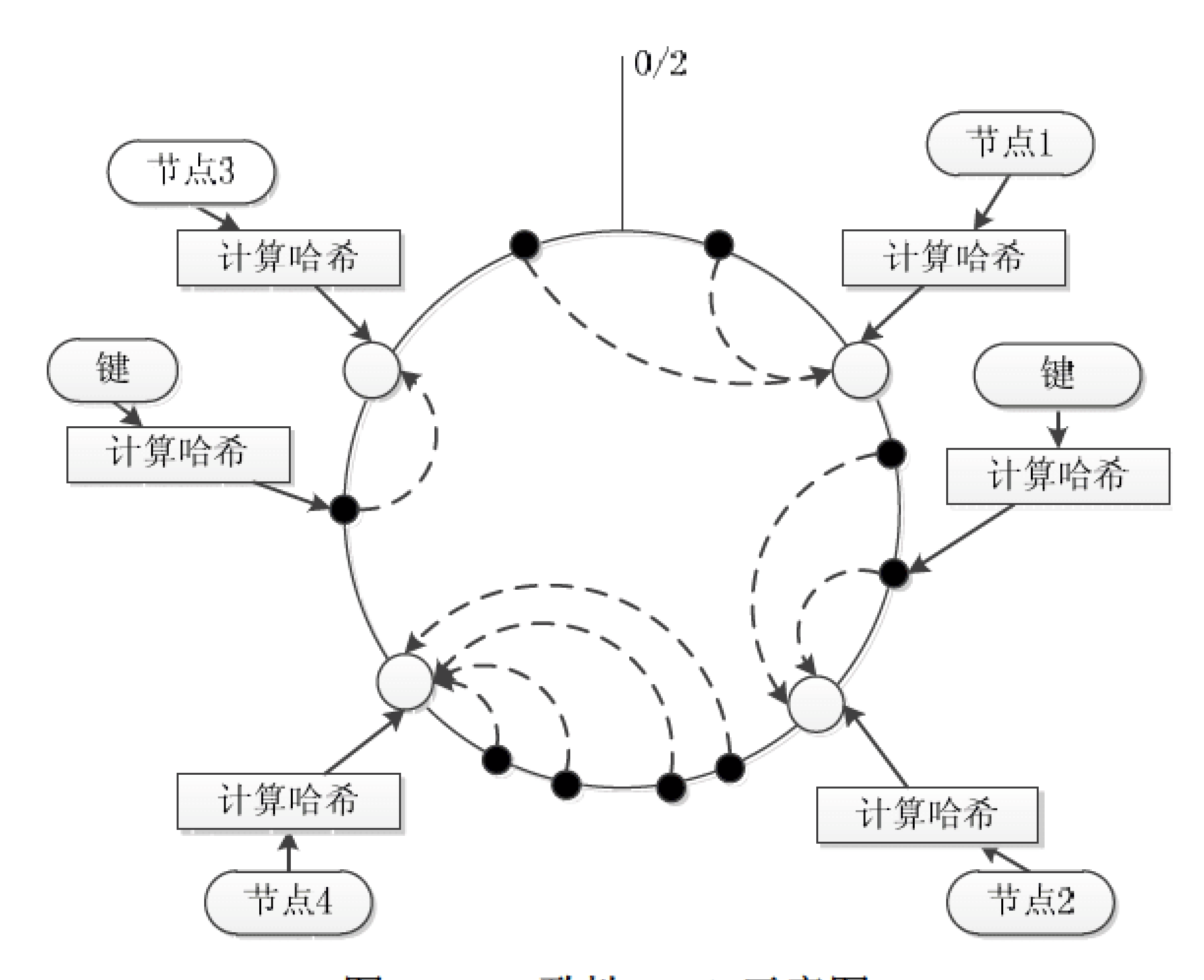


图 2.8　一致性 Hash 示意图

- 通过数据的哈希键值直接定位到相应的存储节点，实现了数据的自动划分。
- 满足系统扩展性需求，最大限度地抑制了节点变化（添加/移除）时重新分布的数据总量。如图 2.9 所示，在节点 5 加入系统前，节点 4 存储映射到区间（节点 2，节点 4）的数据。当系统访问压力过大时，通过增加新的节点 5 来缓解压力。而此时哈希环上只有节点 4

的管理范围——区间（节点 5，节点 4）发生改变，映射到区间（节点 2，节点 5）的数据则由节点 5 存储，因而只需将节点 4 中映射到区间（节点 2，节点 5）的数据（图中椭圆网影区域）迁移到节点 5 中即完成了节点扩展的操作。这避免了大量数据进行迁移的开销，有利于改进系统的可扩展性。

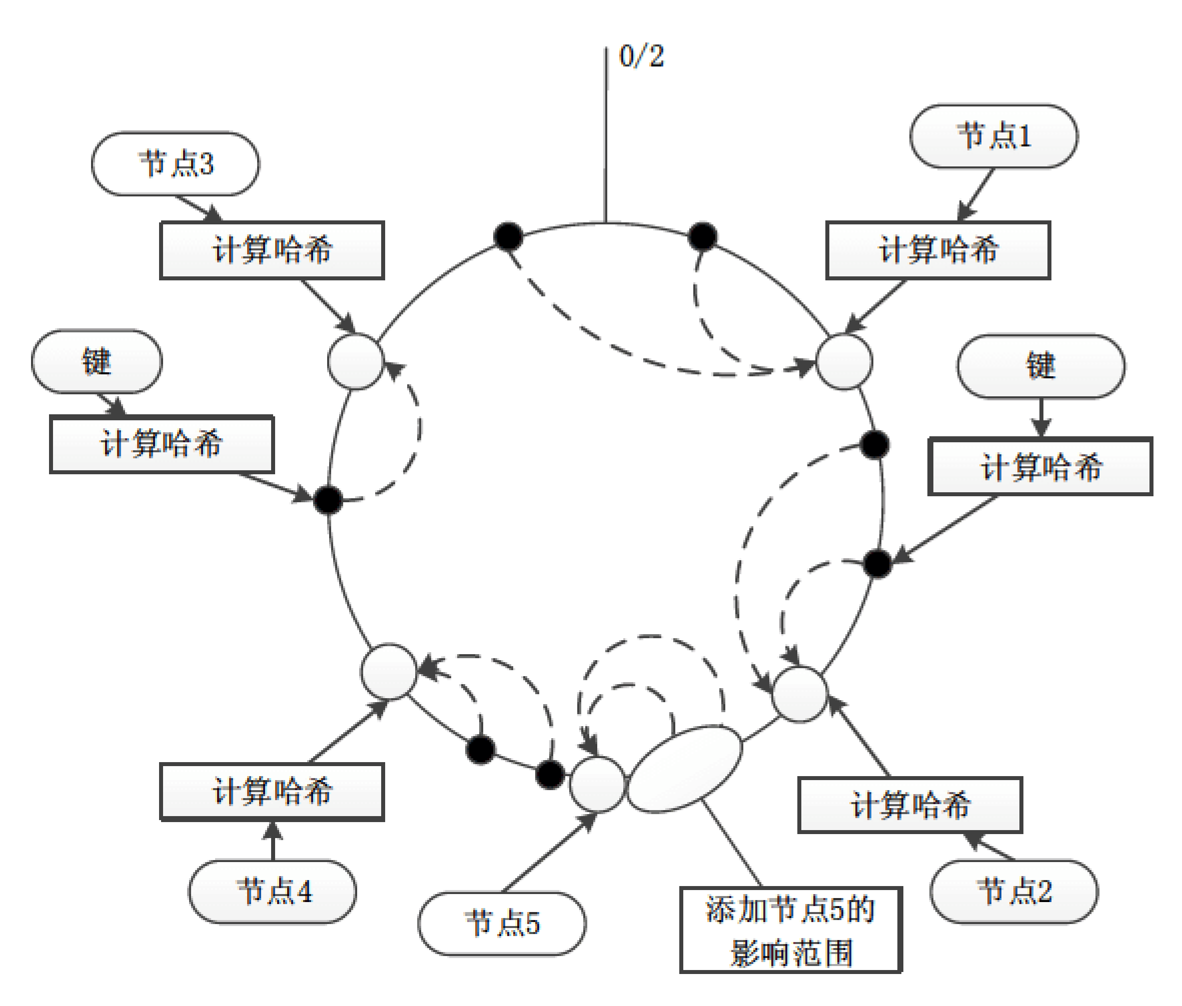


图 2.9　一致性 Hash 加入新结点

由于 Dynamo 采用非集中的管理方式，所以系统中没有专门节点进行元数据信息（数据存放位置等）的存储和管理。数据到存储节点的映射通过系统中各节点在本地维护的存储节点列表进行管理，并利用 gossip 机制进行交互更新。因此针对某一个键的读写操作，每个节点根据本地存放的相关信息，快速定位到正确的节点集执行。

2. 负载均衡

传统的一致性哈希算法在某种程度上解决了系统扩展性的问题，但是无法解决用户访问的随机性以及节点的异构特性所带来的负载不均衡。Dynamo 利用虚拟节点技术，有效地将数据均匀存储到各个节点上，将访问请求的压力分散开，保证了系统的健壮性和负载的均衡性。

虚拟节点的概念是对一致性哈希算法的扩充，它将一个物理节点拆分成多个虚拟节点映射到哈希环上的不同位置，取代了传统一致性哈希中一个物理节点只对应哈希环上一个点的映射关系。使用虚拟节点的优势主要体现在以下方面。

（1）解决节点异构带来的负载不均衡

虚拟节点作为一个资源容器，而存储作为一个服务运行于其中。通过虚拟节点将资源管理粒度单元化。这样，资源多的节点可以多部署一些虚拟节点，而资源少的节点可以少部署一些虚拟节点，从而达到一种相对均衡的状态。

（2）解决用户访问随机性带来的访问压力不均衡

通过将访问压力较大的虚拟节点分配给服务能力强的物理节点进行服务；将访问压力较小的虚拟节点成组分配给服务能力强的物理节点或者逐一分配给服务能力弱的物理节点；最终，达到

动态的负载均衡。

（3）解决用户请求随机性或者节点加入离开时带来的数据不均衡

虚拟节点方法能在最大程度上降低为了数据均衡进行数据迁移所带来的系统开销。比如，当在哈希环中加入一个新的物理节点时，为了保持数据均匀分布的特性，传统的一致性哈希算法需要进行全环节点的数据迁移来到达到数据均衡，这样大大增加了网络的开销。而采用虚拟节点的方法，一个物理节点对应哈希环上的多个虚拟节点，且虚拟节点分布在环中各个位置，进行数据均衡的时候，会从其他物理节点接受总量相当的负载，并且只涉及全环部分虚拟节点的数据迁移，可以在减少迁移数据量的条件下，达到各物理节点负载均衡的目的。

3. 容错以及数据的高可用

Dynamo 通过对数据进行冗余存放来提高数据访问的并发性和保证系统的高可用；通过基于 gossip 机制的异步复制技术完成副本同步；通过客户端采用 Quorum 算法解决多副本带来的一致性问题。即使用所谓的（N，R，W）模型，其中 N 代表系统中每条记录的副本数，W 代表每次记录成功的写操作最少需要参与的副本数，R 代表每次记录成功的读请求最少需要读取的副本数。保证 W+R>N 就能够保证数据的一致性。因为 W+R>N 时读写总会有交集，最少有 W+R－N 个读请求落到被写的副本上，可以利用向量时钟（vector clock）技术，跟踪数据记录在不同机器上的版本变化，以确保当多个读请求结果返回不一致时，能够根据版本信息得出正确的结果。Dynamo 的这种做法是一种折衷，即为了同时保证读和写的效率，写操作不要求绝对同步，而把不同步可能产生的后果推给了读操作。

Dynamo 的副本读写策略非常灵活，针对不同服务的 SLA 需求，提供不同程度的定制策略。通过配置参数 N、W 和 R 以满足不同访问需求的各种场景。比如对于写多读少的操作，可将 W 配低，R 配高；对于写少读多的操作，则可将 W 配高，R 配低。提供低延时读操作的同时，保证用户请求“总是可写”。

为了保证系统容错性能，系统中各节点通过 gossip 机制进行交互以确认节点的活动状态，进行失效侦测，利用多副本机制保证系统可用性。对于临时失效和永久失效的情况，Dynamo 采用不同的策略来容忍失效的发生。在临时失效（网络分区等）发生时，系统通过寻找一台可用节点，将数据临时写在其上，待故障恢复后，临时表中的数据会自动写回原目的地。这样，当临时故障出现时，保证用户总处于可写的状态。而对于永久失效（比如磁盘损坏等），则需要通过副本进行数据恢复。Dynamo 利用 Merkle 树来保证节点失效后副本的同步：系统中每个节点都为每个键范围（key range）维护一个独立的 Merkle 树，当两个节点不一致时（如一个节点发生过一段时间的死机），通过 gossip 机制对比各自的 Merkle 树，快速定位不一致的数据项来进行数据同步。

4. 写操作高可用

Dynamo 中，最重要的是要保证写操作的高可用性，即 Always Writeable，这样就不可避免地牺牲掉数据的一致性。如上所述，Dynamo 中并没有对数据做强一致性要求，而是采用最终一致性。结合上述的数据副本技术，可以看出，若不保证各个副本的强一致性，则用户在读取数据的

时候很可能读到的不是最新的数据。Dynamo 中将数据的增加或删除这种操作都视为一种增加操作，即每一次操作的结果都作为一份全新的数据保存，这样也就造成了一份数据会存在多个版本，分布在不同的节点上。这种情况类似于版本管理中的多份副本同时有多人在修改。多数情况下，系统会自动合并这些版本，一旦合并尝试失败，那么冲突就交给应用层来解决。这时系统表现出来的现象就是，一个 GET（KEY）操作，返回的不是单一的数据，而是一个多版本的数据列表，由用户决定如何合并。这其中的关键技术就是 Vector Clock。

一个 Vector Clock 可以理解为一个<节点编号，计数器>对的列表。每一个版本的数据都会带上一个 Vector Clock。通过对比两份不同数据的 Vector Clock 就能发现它们的关系。所以应用层在读取数据的时候，系统会连带这 Vector Clock 一同返回；在操作数据的时候也需要带上数据的 Vector Clock 一同提交。大致流程如图 2.10 所示。

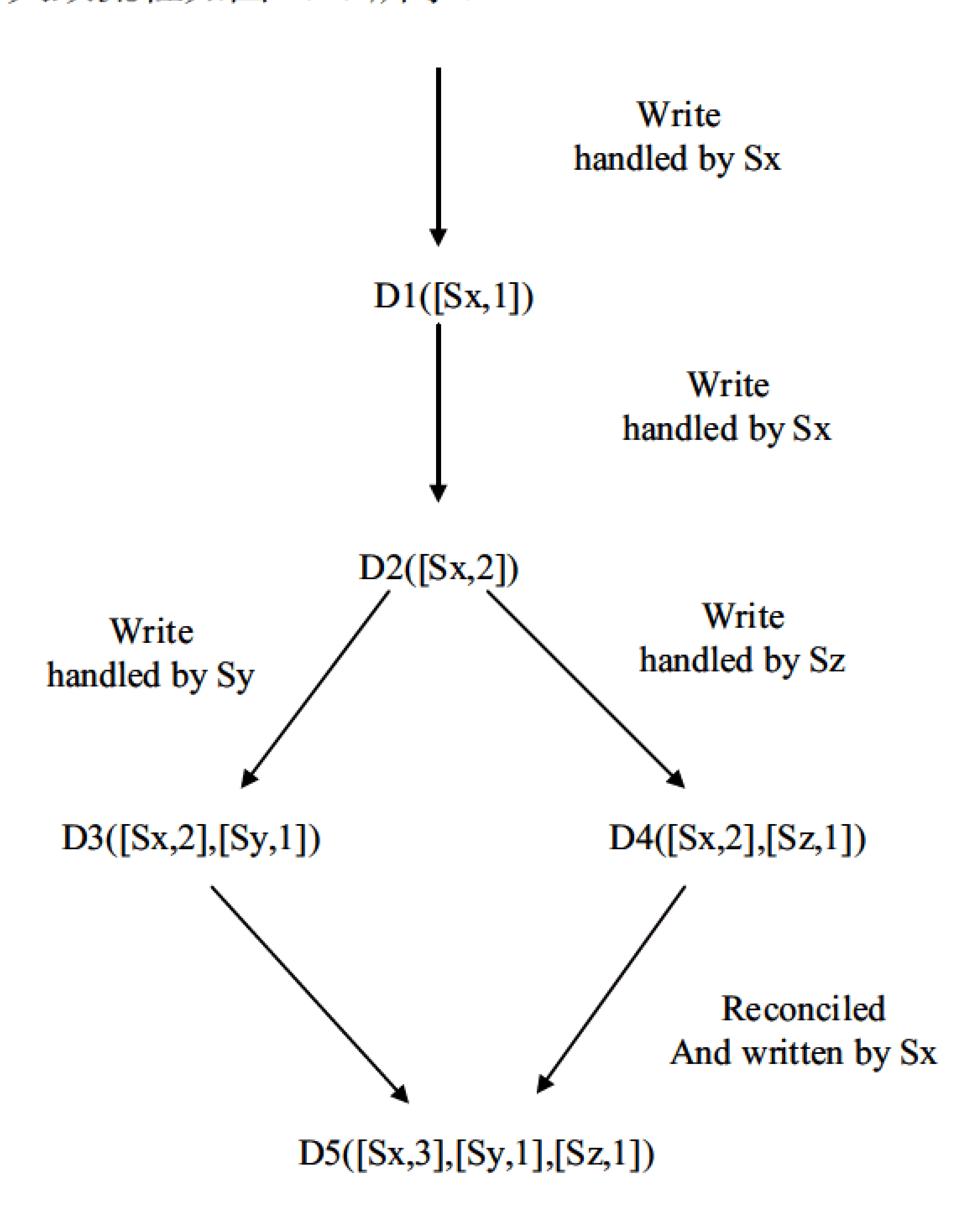


图 2.10　Vector Clock 流程图

5. 一个正常的 GET 和 SET 流程

之前也说过，Amazon 在设计组件的时候采用的是“一切皆服务”的原则。Dynamo 也不例外，应用可以通过两种方式使用 Dynamo：1）通过一个应用路由层转发请求到对应 Dynamo 节点。2）通过连接 Dynamo 的 API 到自己程序中使用。两种方式各有优劣。负责处理读写请求的 Dynamo 节点在系统中称为 Coordinator，一般是之前提到的 Preference List 中的第一台节点。如果不再这个 List 中的机器收到了读写请求，它会将这个请求转发到在 List 中的机器上。

Dynamo 中一共涉及三个重要的参数，其中 N，代表数据的副本数，之前已经说过了。另外还有 W 和 R，分别表示：一次写操作，最小必须写成功节点数；一次读操作，最小读成功节点数。

另外要求 W + R > N。设计这两个参数主要是出于性能考虑，不然可以直接令 W = R = N。读数据时，只要有除了 Coodinator 之外的 R-1 个节点返回了数据，就算是读成功（此时可能返回多个版本的数据）。同理，写数据时，只要有除了 Coordinator 之外的 W-1 个节点写入成功，就算数据写入成功。

6. 故障处理

故障检测和处理往往都是分布式系统的重点和难点，尤其是对于像 Dynamo 这种对可用性要求很高的系统。上边说了，通过设定 N、W 和 R 参数来保证读写的正确。一旦出现读写失败的情况，都会触发故障处理机制。Dynamo 中将故障分为两类，一类是临时性的故障，另一类是持久的故障，分别对应两种不同的处理方式。如图 2.11 所示。

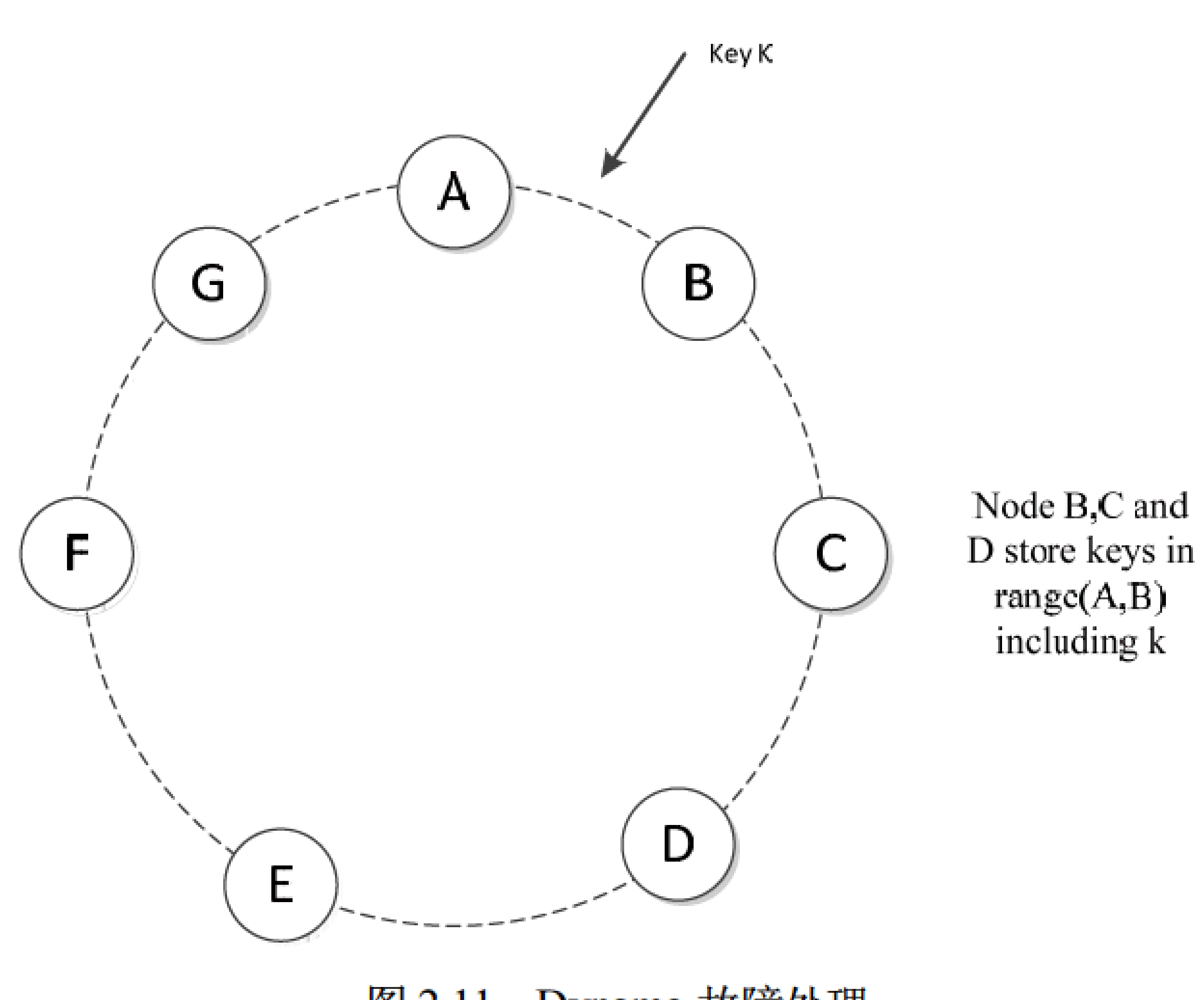


图 2.11　Dynamo 故障处理

以 N=3 为例，如果在一次写操作时发现节点 A 宕机了，那么本应该存在 A 上的副本就会发送到 D 上，同时在 D 中会记录这个副本的元信息（MetaData）。其中有个标示表明这份数据是本应该存在 A 上的，一旦节点 D 之后检测到 A 从故障中恢复了，D 就会将这个本属于 A 的副本回传给 A，之后删除这份数据。Dynamo 中称这种技术为 Hinted Handoff。另外为了应对整个机房掉线的故障，Dynamo 中应用了一个很巧妙的方案。之前说过 Preference List，每次读写都会从这个列表中取出 R 或 W 个节点。那么只要在这个列表生成的时候，让其中的节点是分布于不同机房的，数据自然就写到了不同机房的节点上。

Hinted Handoff 的方式在少量的或是短暂的机器故障中表现很好，但是在某些情况下仍然会导致数据丢失。如上所说，如果节点 D 发现 A 重新上线了，会将本应该属于 A 的副本回传过去，这期间 D 发生故障就会导致副本丢失。为了应对这种情况，Dynamo 中应用了基于 Merkle Tree 的 Anti-Entrpy 系统。

2.4.3 BigTable 的开源实现 HBase

HBase 是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价PC Server上搭建起大规模结构化存储集群。HBase是Google Bigtable的开源实现，类似Google Bigtable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统，如图 2.12 所示。Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。图 2.12 描述了 Hadoop EcoSystem 中的各层系统，其中 HBase 位于结构化存储层，Hadoop HDFS 为 HBase 提供了高可靠性的底层存储支持，Hadoop MapReduce 为 HBase 提供了高性能的计算能力，Zookeeper 为 HBase 提供了稳定服务和 failover 机制。

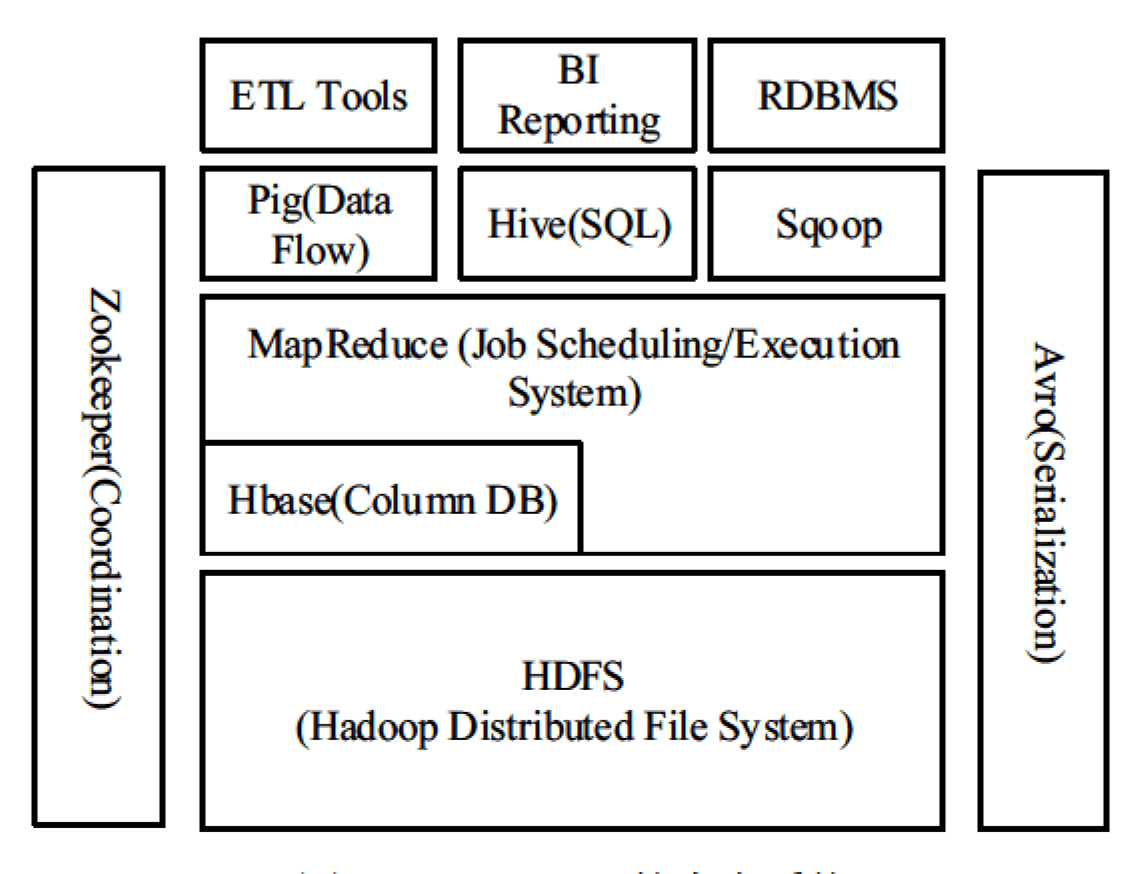


图 2.12　Hadoop 的生态系统

此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变得非常简单。Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库数据向 HBase 中迁移变得非常方便。

1. HBase 的集群架构

由于安装 HBase 是需要 Hadoop 环境的，因此 HBase 事实上是搭建在 Hadoop 集群中的。事实上，HBase 中的数据是以文件的形式存在于 HDFS 中的，从集群结构上讲，也是主从结构，由一个主服务器和多个从属服务器组成。

如图 2.13 为 HBase 的集群组成。前面已经介绍过，在 HBase 中，主服务器叫做 Master，从属服务器命名为 RegionServer。Master 是整个系统的入口，系统启动时会由 Master 将数据库中表的子表 Region 分配到各个 RegionServer 上，并维护 Region 分配表，当客户端的请求到达时，由 Master 处理并分配到具体的 RegionServer，在此过程中，Master 会保持与各个 RegionServer 的连接，并监控其运行情况，在分配客户端请求时保持集群负载平衡。HRegionServer 则管理着一组 Region 的对象 HRegion，并直接处理来自客户端的读写操作等。

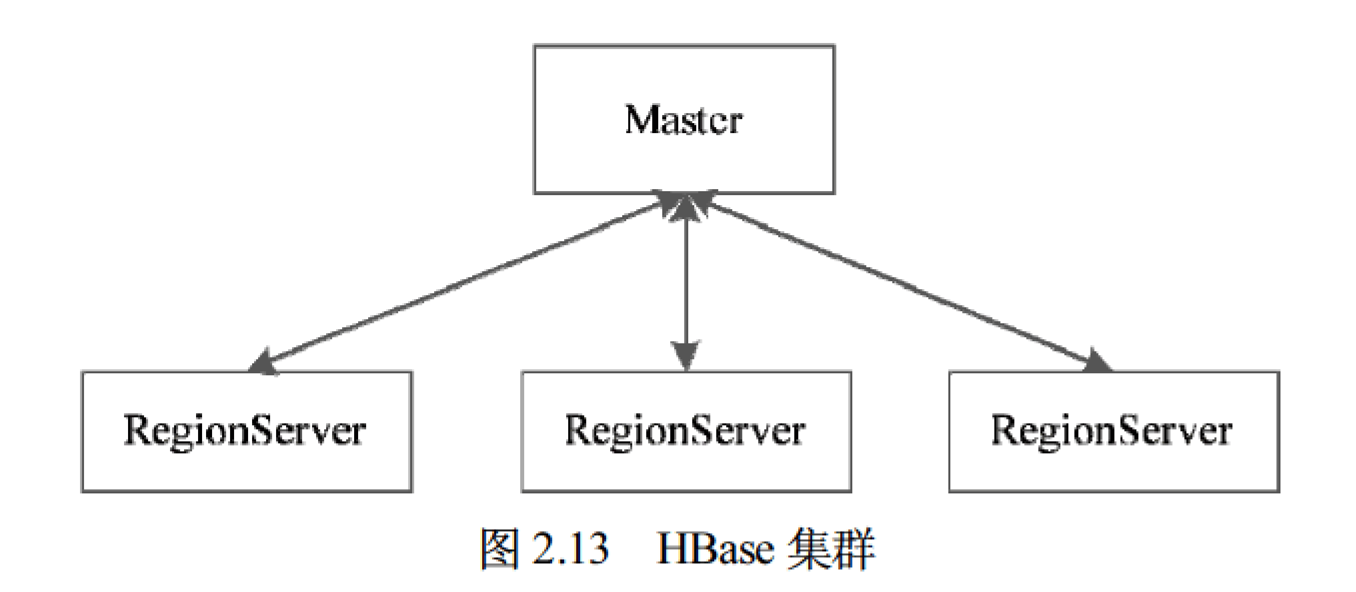


图 2.13　HBase 集群

2. HBase 的系统架构

HBase 的集群并不是只有 Master 和 RegionServer 就能工作的，为了支撑 HBase 的正常运行，事实上 HBase 也有它自己的生态群。读者可以通过划分层次的方式进一步认识 HBase 的系统架构。

图 2.14 给出了 HBase 的系统架构。HBase 的集群结构处于服务层，HBase 客户端处于应用层，而分布式文件系统和分布式锁服务则处于支撑层。从各层关系来看，支撑层向服务层提供必要的底层服务支持，例如 HDFS 向 HRegionServer 提供数据持久化，分布式锁机制则协调集群调度、管理 HMaster 节点等；服务层则向应用层提供业务逻辑处理服务，HBase 向客户端提供的服务调用接口主要是客户端 API；应用层的客户端可以调用客户端 API 实现特定应用。

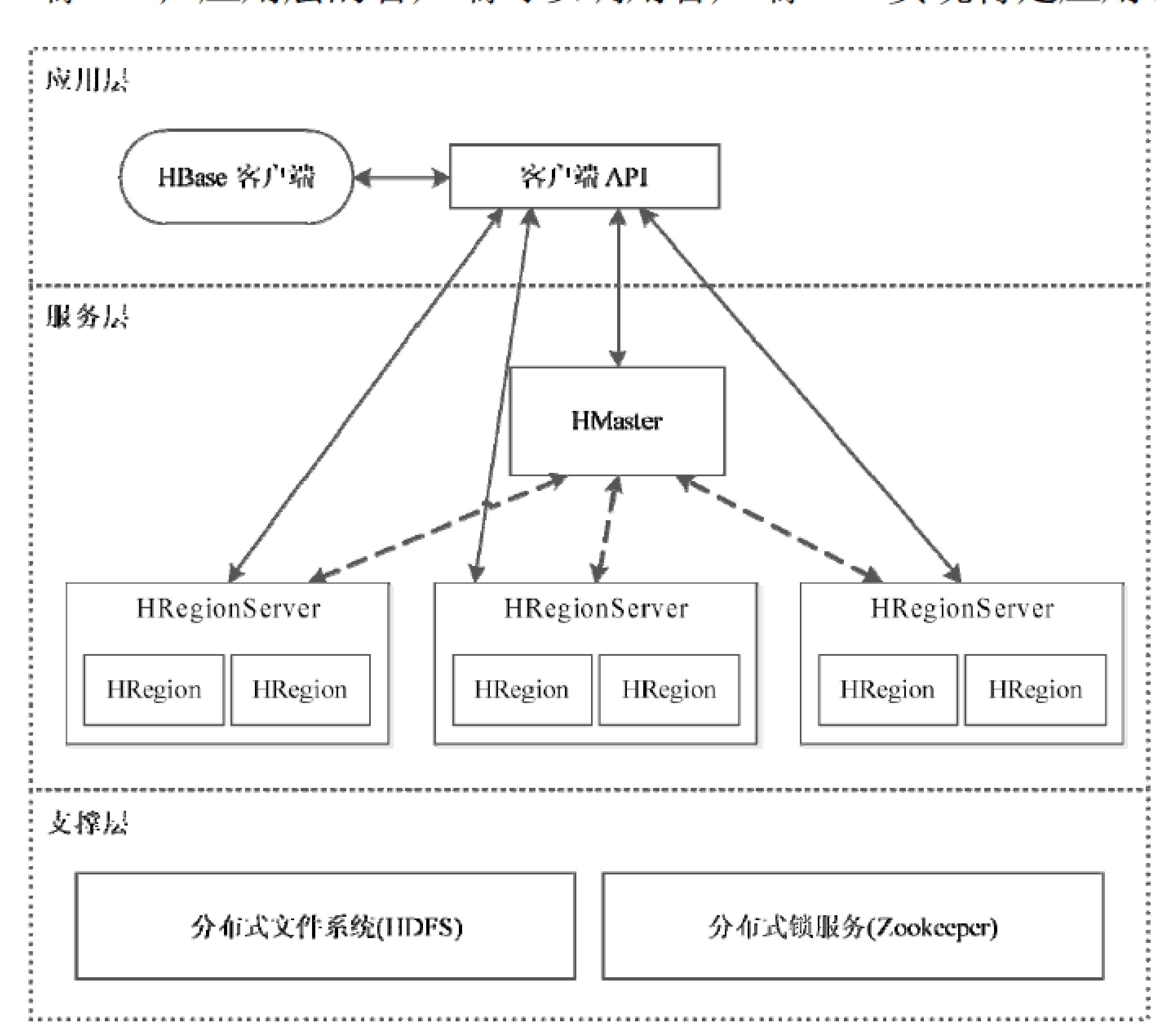


图 2.14　HBase 的系统架构

细心的读者会发现该图中没有出现元数据表-ROOT-和.META.，其实这两个表也是在 HBase 集群启动时分配的，它们也被当做 Regions 分配到 HRegionServer 来管理，所以没有特别标注出来。

支撑层的 HDFS 对于 HBase 而言是不可或缺的，HBase 本身不会存储数据，表中的数据都需要以文件的方式持久化到分布式文件系统中。而 ZooKeeper 主要是为了解决 HMaster 的单点失效问题，当启动多个 HMaster 时，由 ZooKeeper 保证总有一个 Master 在运行，因此没有配置 ZooKeeper 的 HBase 仍然是可以使用的，但要明确认识到单点失效所带来的风险。

2.4.4 MongoDB

MongoDB 是一个基于分布式文件存储的数据库，由 C++语言编写。旨在为 Web 应用提供可扩展的高性能数据存储解决方案。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富、最像关系数据库的。它支持的数据结构非常松散，是类似 json 的 bson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

MongoDB 主要特性有：面向文档存储、支持完全索引、高可靠性、自动分片支持云级扩展性、查询记录分析、快速就地更新、支持 Map/Reduce、提供分布式文件系统 GridFS、商业支持。除此之外，MongoDB 还具有模式自由、支持动态查询、支持数据复制与故障恢复、使用高效的二进制数据存储、包括大型对象、支持非常多的语言、文件存储格式为 BSON。

MongoDB 的主要目标是在键/值存储方式与传统的 RDBMS 系统之间架起一座桥梁，集两者的优势于一身。MongoDB 适合用于网站数据：MongoDB 非常适合实时的插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性；MongoDB 适合用于缓存：由于性能很高，MongoDB 也适合作为信息基础设施的缓存层，在系统重启之后，由 MongoDB 搭建的持久化缓存层可以避免下层的数据源过载；MongoDB 适合用于大尺寸、低价值的数据，使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储；MongoDB 适合用于高伸缩性的场景，MongoDB 非常适合由数十或数百台服务器组成的数据库。MongoDB 的路线图中已经包含对 MapReduce 引擎的内置支持；用于对象及 JSON 数据的存储，MongoDB 的 BSON 数据格式非常适合文档化格式的存储及查询。

MongoDB 的使用也会有一些限制。MongoDB 不适合高度事务性的系统，例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。MonogoDB 不适用于传统的商业智能应用，针对特定问题的 BI 数据库会产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。

1. MongoDB 数据文件内部结构

MongoDB 在数据存储上按命名空间来划分，一个 Collection 是一个命名空间，一个索引也是一个命名空间。如图 2.15 所示。

- 同一个命名空间的数据被分成很多个 Extent，Extent 之间使用双向链表连接。
- 在每一个 Extent 中，保存了具体每一行的数据，这些数据也是通过双向链接来连接的。
- 每一行数据存储空间不仅包括数据占用空间，还可能包含一部分附加空间，这使得在数据 Update 变大后可以不移动位置。
- 索引以 BTree 结构实现。

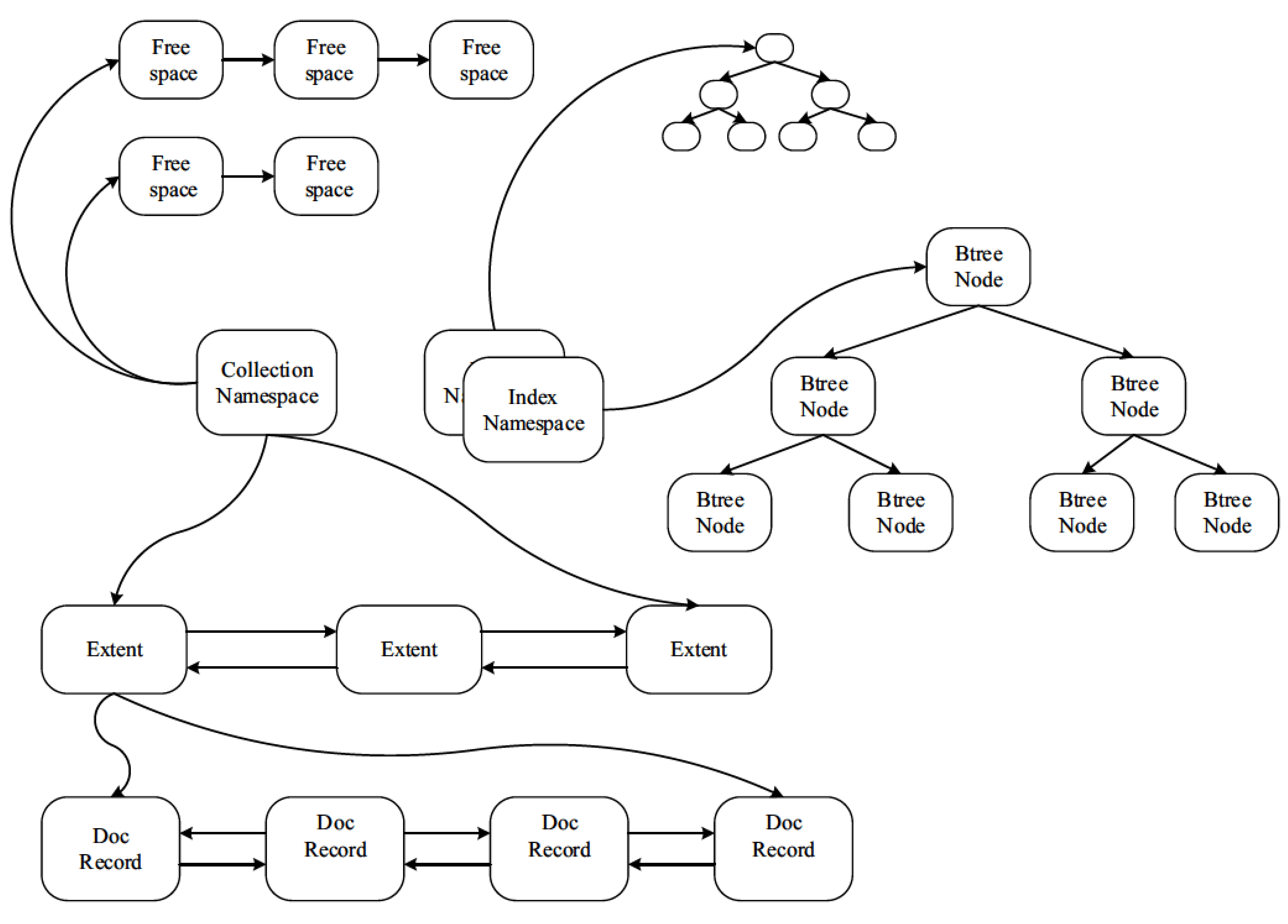


图 2.15　MongoDB 文件内部结构图

2. MongoDB 数据同步

MongoDB 采用 Replica Sets 模式的同步流程，如图 2.16 所示，本流程可简要描述如下：

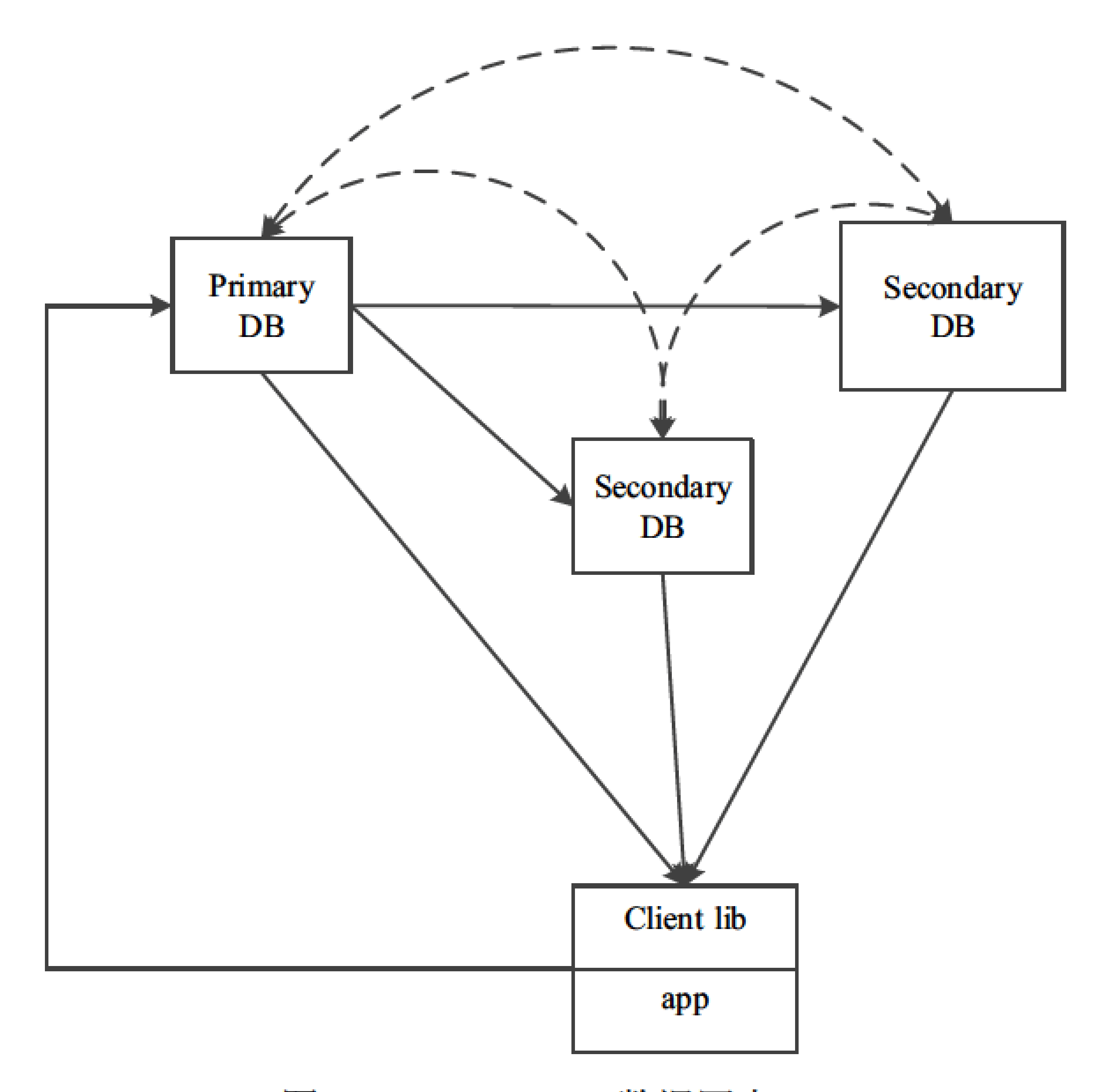


图 2.16　MongoDB 数据同步

黑色虚线箭头表示写操作可以写到 Primary 上，然后异步同步到多个 Secondary 上。黑色实线箭头表示读操作可以从 Primary 或 Secondary 任意一个中读取。各个 Primary 与 Secondary 之间一直保持心跳同步检测，用于判断 Replica Sets 的状态。

如图 2.17 所示，MongoDB 的分片是指定一个分片 key 来进行，数据按范围分成不同的 chunk，每个 chunk 的大小有限制。有多个分片节点保存这些 chunk，每个节点保存一部分的 chunk。每一个分片节点都是一个 Replica Sets，这样可保证数据的安全性。当一个 chunk 超过其限制的最大体积时，会分裂成两个小的 chunk。当 chunk 在分片节点中分布不均衡时，会引发 chunk 迁移操作。

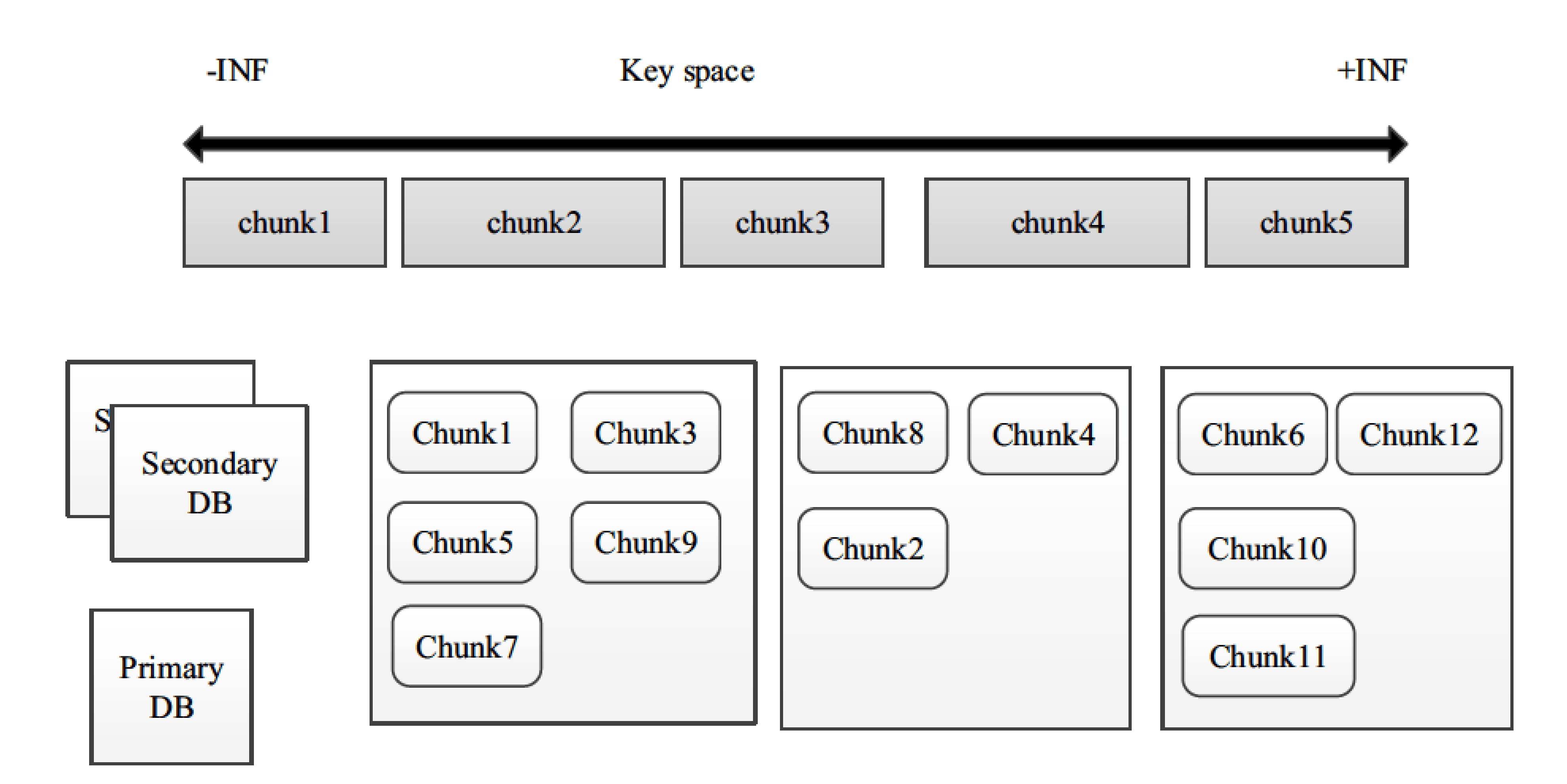


图 2.17　MongoDB 分片

3. 服务器角色

前面讲了分片的机制，下面是具体在分片时几种节点的角色，如图 2.18 所示。

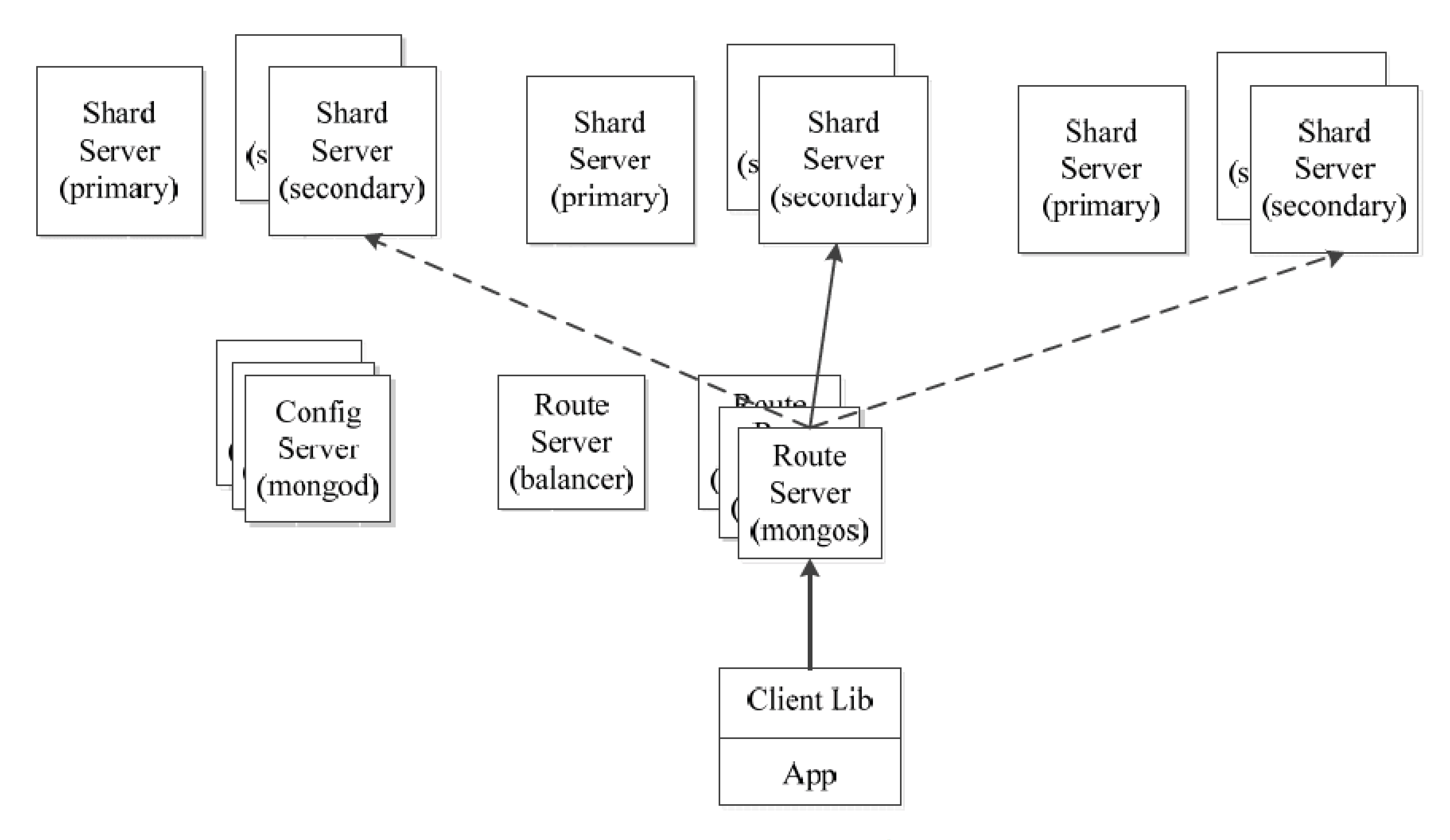


图 2.18　MongoDB 分片机制

客户端访问路由节点 mongos 来进行数据读写。config 服务器保存了两个映射关系，一个是

key 值的区间对应哪一个 chunk 的映射关系，另一个是 chunk 存在哪一个分片节点的映射关系。路由节点通过 config 服务器获取数据信息，通过这些信息，找到真正存放数据的分片节点进行对应操作。路由节点还会在写操作时判断当前 chunk 是否超出限定大小。如果超出，就分列成两个 chunk。对于按分片 key 进行的查询和 update 操作来说，路由节点会查到具体的 chunk 然后再进行相关的工作。对于不按分片 key 进行的查询和 update 操作来说，mongos 会对所有下属节点发送请求然后再对返回结果进行合并。

2.4.5　CouchDB

CouchDB 是一个开源的面向文档的数据库管理系统，可以通过 RESTful JavaScript Object Notation（JSON）API 访问。术语“Couch”是 “Cluster Of Unreliable Commodity Hardware”的首字母缩写，它反映了 CouchDB 的目标具有高度可伸缩性，提供了高可用性和高可靠性，即使运行在容易出现故障的硬件上也是如此。CouchDB 最初是用 C++编写的，但在 2008 年 4 月，这个项目转移到 Erlang OTP 平台进行容错测试。

CouchDB 是用 Erlang 开发的面向文档的数据库系统。CouchDB 不是一个传统的关系数据库，而是面向文档的数据库，其数据存储方式有点类似 Lucene 的 index 文件格式，CouchDB 最大的意义在于它是一个面向 Web 应用的新一代存储系统，事实上，CouchDB 的口号就是：下一代的 Web 应用存储系统。

CouchDB 有以下特点：

- CouchDB 是分布式的数据库，它可以把存储系统分布到 n 台物理的节点上面，并且很好地协调和同步节点之间的数据读写一致性。这当然也得靠 Erlang 无与伦比的并发特性才能做到。对于基于 Web 的大规模应用文档应用，分布式可以让它不必像传统的关系数据库那样分库拆表，在应用代码层进行大量的改动。
- CouchDB 是面向文档的数据库，存储半结构化的数据，比较类似 lucene 的 index 结构，特别适合存储文档，因此很适合 CMS、电话本、地址本等应用，在这些应用场合，文档数据库要比关系数据库更加方便，性能更好。
- CouchDB 支持 REST API，可以让用户使用 JavaScript 来操作 CouchDB 数据库，也可以用 JavaScript 编写查询语句，我们可以想像一下，用 AJAX 技术结合 CouchDB 开发出来的 CMS 系统会是多么简单和方便。

其实 CouchDB 只是 Erlang 应用的冰山一角，在最近几年，基于 Erlang 的应用也得到的蓬勃的发展，特别是在基于 Web 的大规模、分布式应用领域，几乎都是 Erlang 的优势项目。CouchDB 的架构图如图 2.19 所示。

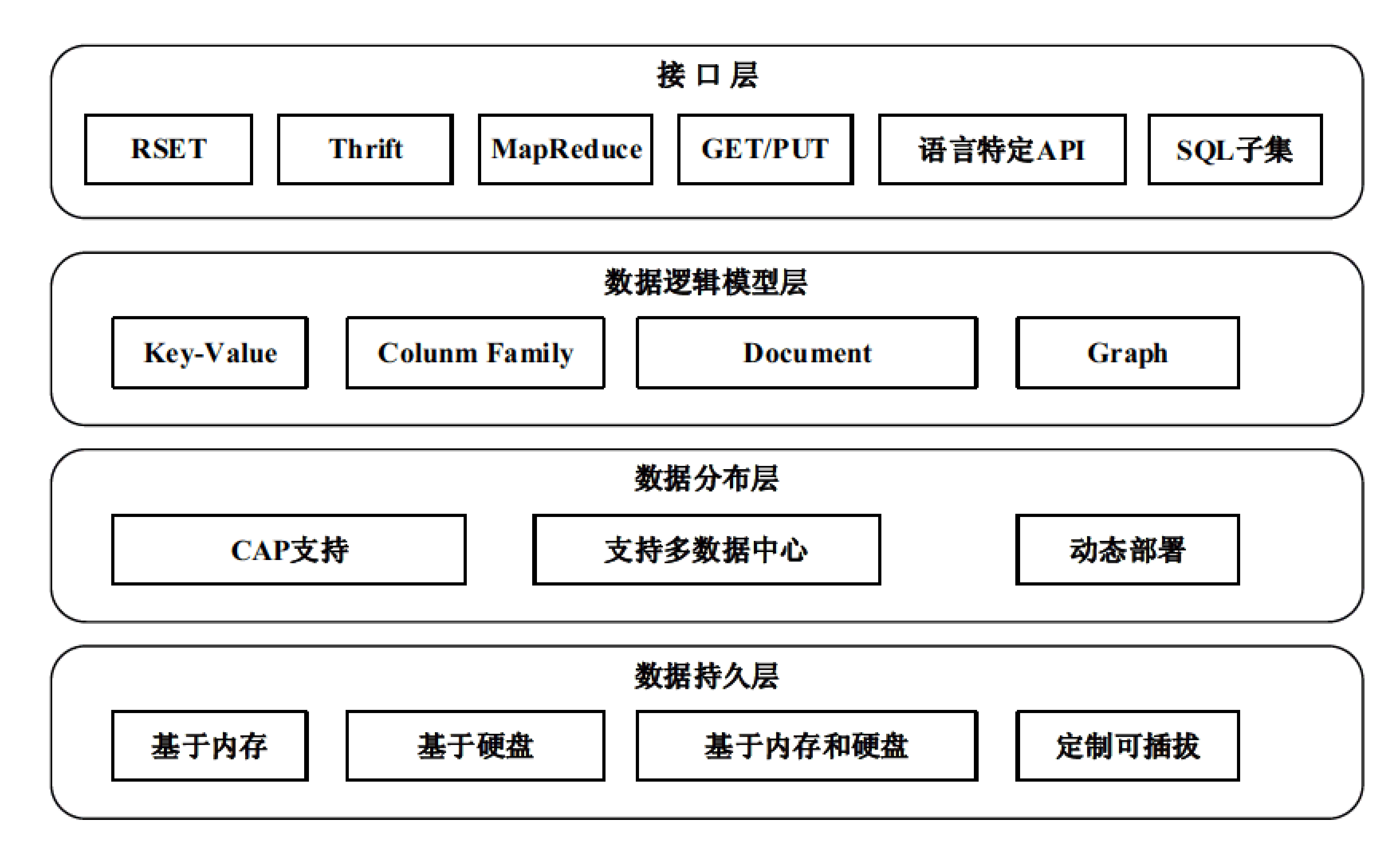


图 2.19 CouchDB 的架构图

CouchDB 构建在强大的 B-树储存引擎之上。这种引擎负责对 CouchDB 中的数据进行排序，并提供一种能够在对数均摊时间内执行搜索、插入和删除操作的机制。CouchDB 将这个引擎用于所有内部数据、文档和视图。

因为 CouchDB 数据库的结构独立于模式，所以它依赖于使用视图创建文档之间的任意关系，以及提供聚合和报告特性。使用 Map/Reduce 计算这些视图的结果，Map/Reduce 是一种使用分布式计算来处理和生成大型数据集的模型。Map/Reduce 模型由 Google 引入，可分为 Map 和 Reduce 两个步骤。在 Map 步骤中，由主节点接收文档并将问题划分为多个子问题。然后将这些子问题发布给工作节点，由它处理后再将结果返回给主节点。在 Reduce 步骤中，主节点接收来自工作节点的结果并合并它们，以获得能够解决最初问题的总体结果和答案。

CouchDB 中的 Map/Reduce 特性生成键/值对，CouchDB 将它们插入到 B-树引擎中并根据它们的键进行排序。这就能通过键进行高效查找，并且提高 B-树中的操作的性能。此外，这还意味着可以在多个节点上对数据进行分区，而不需要单独查询每个节点。

传统的关系数据库管理系统有时使用锁来管理并发性，从而防止其他客户机访问某个客户机正在更新的数据。这就防止多个客户机同时更改相同的数据，但对于多个客户机同时使用一个系统的情况，数据库在确定哪个客户机应该接收锁并维护锁队列的次序时会遇到困难，这很常见。在 CouchDB 中没有锁机制，它使用的是多版本并发性控制（Multiversion Concurrency Control，MVCC），向每个客户机提供数据库的最新版本的快照。这意味着在提交事务之前，其他用户不能看到更改。许多现代数据库开始从锁机制前移到 MVCC，包括 Oracle（V7 之后）和 Microsoft® SQL Server 2005 及更新版本。

2.4.6 Redis

Redis 是一款开源的、高性能的键-值存储（key-value store）。它常被称作是一款数据结构服务器（data structure server）。Redis 的键值可以包括字符串（strings）、哈希（hashes）、列表（lists）、

集合（sets）和 有序集合（sorted sets）等数据类型。 对于这些数据类型，可以执行原子操作。例如：对字符串进行附加操作（append）；递增哈希中的值；向列表中增加元素；计算集合的交集、并集与差集等。

为了获得优异的性能，Redis 采用了内存中（in-memory）数据集（dataset）的方式。根据使用场景的不同，可以每隔一段时间将数据集转存到磁盘上来持久化数据，或者在日志尾部追加每一条操作命令。

Redis 同样支持主从复制（master-slave replication），并且具有非常快速的非阻塞首次同步（non-blocking first synchronization）、网络断开自动重连等功能。同时 Redis 还具有其他一些特性，其中包括简单的 check-and-set 机制、pub/sub 和配置设置等，以便使得 Redis 能够表现得更像缓存（cache）。

Redis 还提供了丰富的客户端，以便支持现阶段流行的大多数编程语言。详细的支持列表可以参看 Redis 官方文档：http://redis.io/clients。Redis 自身使用 ANSI C 来编写，并且能够在不产生外部依赖（external dependencies）的情况下运行在大多数 POSIX 系统上，如 Linux、*BSD、OS X 和 Solaris 等。

redis 提供的数据类型有：string，hash，list，set 及 zset（sorted set）。

1. string（字符串）

string 是最简单的类型，可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value，其上支持的操作与 Memcached 的操作类似。但它的功能更丰富。redis 采用结构 sdshdr 和 sds 封装了字符串，字符串相关的操作实现在源文件 sds.h/sds.c 中。sdshdr 数据结构定义如下：

```
typedef char *sds;
struct sdshdr {
 long len;
 long free;
 char buf[];
};
```

2. list（双向链表）

list 是一个链表结构，主要功能是 push、pop、获取一个范围的所有值等。操作中 key 理解为链表的名字。对 list 的定义和实现在源文件 adlist.h/adlist.c 中，相关的数据结构定义如下：

```
// list 迭代器
typedef struct listIter {
listNode *next;
int direction;
} listIter;
// list 数据结构
typedef struct list {
```

```
 listNode *head;
 listNode *tail;
 void *(*dup)(void *ptr);
 void (*free)(void *ptr);
 int (*match)(void *ptr, void *key);
 unsigned int len;
 listIter iter;
} list;
```

3. dict（hash 表）

set 是集合，和数学中的集合概念相似，对集合的操作有添加删除元素、对多个集合求交并差等操作。操作中 key 理解为集合的名字。在源文件 dict.h/dict.c 中实现了 hashtable 的操作，数据结构的定义如下：

```
// dict 中的元素项
typedef struct dictEntry {
 void *key;
 void *val;
 struct dictEntry *next;
 } dictEntry;
// dict 相关配置函数
typedef struct dictType {
 unsigned int (*hashFunction)(const void *key);
 void *(*keyDup)(void *privdata, const void *key);
 void *(*valDup)(void *privdata, const void *obj);
 int (*keyCompare)(void *privdata, const void *key1, const void *key2);
 void (*keyDestructor)(void *privdata, void *key);
 void (*valDestructor)(void *privdata, void *obj);
} dictType;
// dict 定义
typedef struct dict {
 dictEntry **table;
 dictType *type;
 unsigned long size;
 unsigned long sizemask;
 unsigned long used;
 void *privdata;
} dict;
// dict 迭代器
```

```
typedef struct dictIterator {
 dict *ht;
 int index;
 dictEntry *entry, *nextEntry;
} dictIterator;
```

dict 中 table 为 dictEntry 指针的数组，数组中每个成员为 hash 值相同元素的单向链表。set 是在 dict 的基础上实现的，指定了 key 的比较函数为 dictEncObjKeyCompare，若 key 相等则不再插入。

4. zset（排序 set）

zset 是 set 的一个升级版本，它在 set 的基础上增加了一个顺序属性，这一属性在添加修改元素的时候可以指定，每次指定后，zset 会自动重新按新的值调整顺序。可以理解为有两列的 mysql 表，一列存 value，一列存顺序。操作中 key 理解为 zset 的名字。

```
typedef struct zskiplistNode {
 struct zskiplistNode **forward;
 struct zskiplistNode *backward;
 double score;
 robj *obj;
} zskiplistNode;
typedef struct zskiplist {
 struct zskiplistNode *header, *tail;
 unsigned long length;
 int level;
} zskiplist;
typedef struct zset {
 dict *dict;
 zskiplist *zsl;
} zset;
```

zset 利用 dict 维护 key -> value 的映射关系，用 zsl（zskiplist）保存 value 的有序关系。zsl 实际是叉数不稳定的多叉树，每条链上的元素从根节点到叶子节点保持升序排序。

redis 使用了两种文件格式：全量数据和增量请求。全量数据格式是把内存中的数据写入磁盘，便于下次读取文件进行加载；增量请求文件则是把内存中的数据序列化为操作请求，用于读取文件进行 replay 得到数据，序列化的操作包括 SET、RPUSH、SADD、ZADD。

redis 的存储分为内存存储、磁盘存储和 log 文件三部分，配置文件中有三个参数对其进行配置。

- save seconds updates，save 配置，指出在多长时间内，有多少次更新操作，就将数据同步到数据文件。这个可以多个条件配合，比如默认配置文件中的设置，就设置了三个条件。

- appendonly yes/no ，appendonly 配置，指出是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis 本身同步数据文件是按上面的 save 条件来同步的，所以有的数据会在一段时间内只存在于内存中。
- appendfsync no/always/everysec ，appendfsync 配置，no 表示等操作系统进行数据缓存同步到磁盘，always 表示每次更新操作后手动调用 fsync()将数据写到磁盘，everysec 表示每秒同步一次。

2.4.7 Hypertable

Hypertable 是一个正在进行中的开源项目，以 google 的 bigtable 论文为基础指导，使用 C++ 语言实现。

Hypertable 的目标是为了解决大并发，大数据量的数据库需求。目前的 Hypertable 不支持事物，也不支持关联查询，对单条查询的响应时间可能也不如传统数据库。

HyperTable 的优点如下：

- 并发性：可以处理大量并发请求，管理大量数据。
- 规模：可扩缩性好，扩容只需要增加集群中的机器。
- 可用性：任何节点失效，既不会造成系统瘫痪也不会丢失数据。在集群节点足够的情况下，并发量和数据量对性能基本没有影响。

1. Hypertable 的总体架构

图 2.20 是 Hypertable 项目文档中的总体架构图。简单介绍一下各个组成部分。

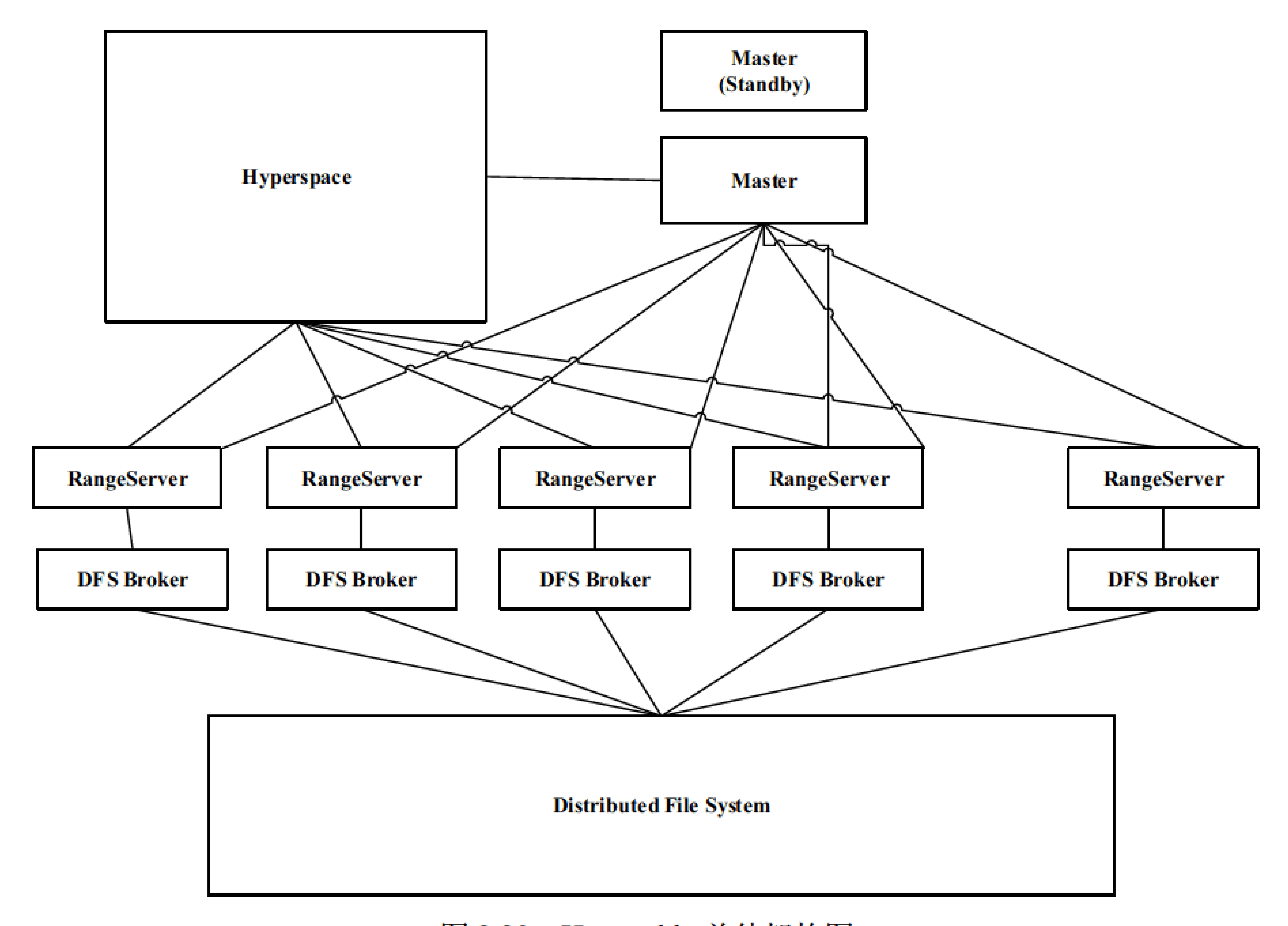


图 2.20　Hypertable 总体架构图

（1）Hyperspace：可以认为是一个文件存储系统，用来存储一些元数据信息，同时提供一些锁的功能。在 google 的 bigtable 论文中，这个模块叫做 Chubby，是采用一个小集群来实现的。在 Hypertable 目前的版本中，这是一台单台服务器，内部采用 Berkeley DB 实现。单点服务器制造了一个系统瓶颈，这台服务器不能失效，所以在将来的版本中，这个模块将采用类似 Chubby 的集群实现。

（2）Master：Master 有两个主要责任。一是管理元操作，比如建表，更改表结构等。二是管理 RangeServer，检测 RangeServer 的工作状态，调整 RangeServer 服务的数据以实现负载均衡。

（3）RangeServer：RangeServer 是真正提供服务的单元，每个 RangeServer 管理一张表的部分数据。在 Hypertable 系统中，表按 row range 分割为许多 tablet，这些 tablet 由 range server 维护，一个 range server 主机可以维护多个 tablet。Rangeserver 负责处理所有的该 row range 的读写操作。

（4）DFS Broker：它的主要作用是使用底层的文件系统来完成 Hypertable 对文件系统的请求。Hypertable 对文件系统的使用有一个很简单的接口，只需要文件系统提供几个很简单的操作。Hypertable 的读写文件等操作都以 socker 形式向 DFS Broker 发出请求来完成。这样对于不同的文件系统，只要实现一个 DFS Broker 就可以与 Hypertable 一起工作了。

（5）存储引擎：Hypertable 设计是为了使用分布式存储系统，但是当然也可以使用本地文件系统。存储引擎只需要提供最简单的几个功能就可以与 Hypertable 协同工作了。

2. 存储结构介绍

考虑到大部分分布式文件系统对文件追加容易修改困难的特点，Hypertable 采用了只追加不修改的文件存储方式。任何对原来记录的修改都是在后面添加一条新的记录。在读取的时候可以指定读取的版本数。当每次只读取最新版本的时候，看到的就是最终结果。一个 range 在存储上是一系列的有序的数据块。Hypertable 称之为 CellStore。

CellStore 是由一些列的 block 再加上一个对 block 的索引以及一些查询优化信息（Bloom Filter）组成，如图 2.21 所示。

首先，一个 CellStore 中的数据是按照主键排序的，其次，索引是只索引到 block 的。举一个具体的例子来看一下（我们只关注 block 和索引，忽略 Bloom Filter 信息文件头信息等）。还是那个 id，name 两个列的表。我们现在看一个 CellStore，id 从 0 到 400。每一百数据为一个 block。那么存储起来好像是图 2.21 所示的样子。

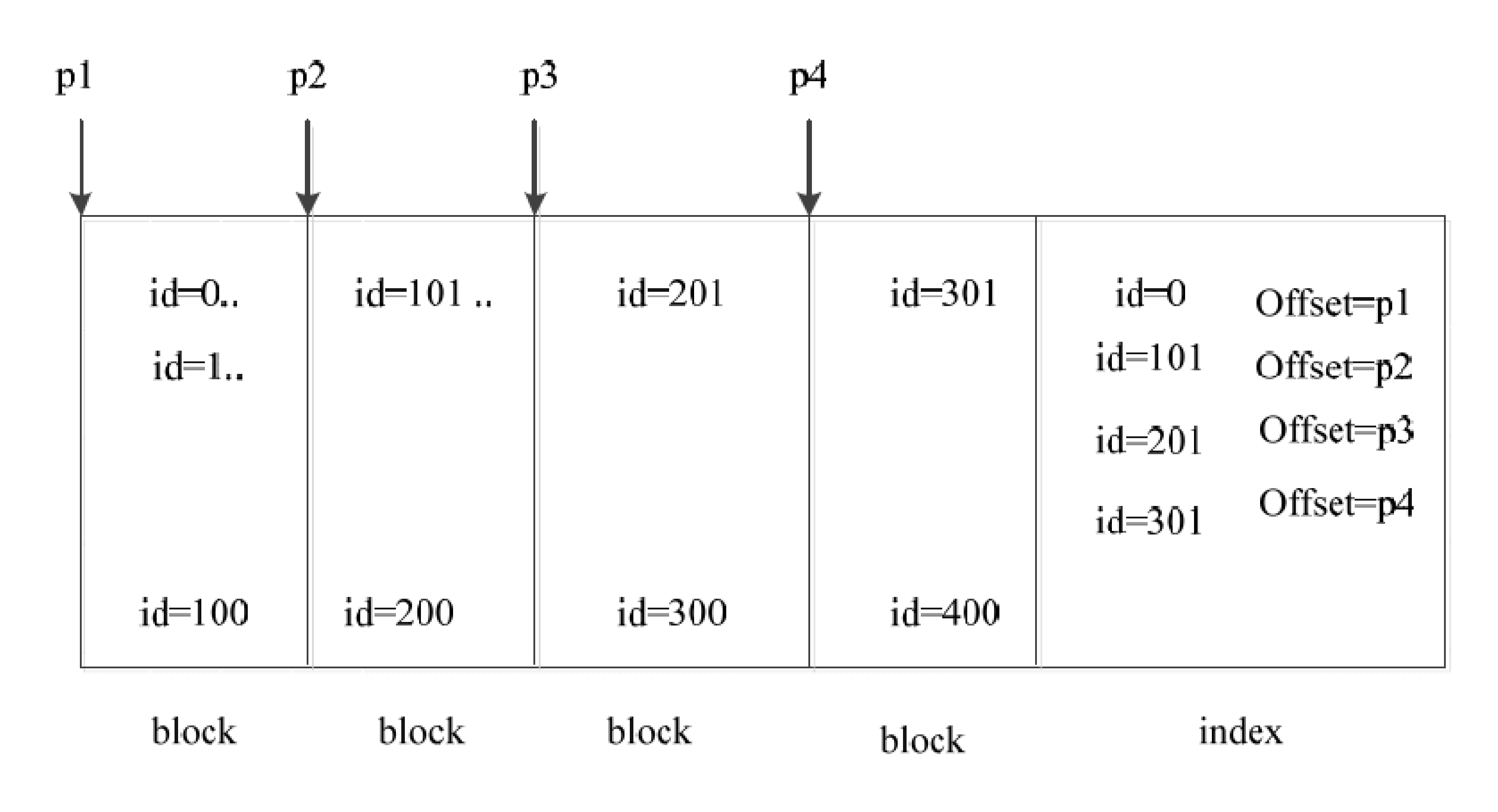


图 2.21　CellStore 示意图

一个 range 收到查询请求的时候先读取 CellStore 中的索引部分，这样可以定位出所需要的数据在哪个 block 中，然后读取相应的 block 并遍历查找所需要的数据。一般一个 block 的大小为 64KB。因为所有插入操作其实都是在内存中进行。当内存中数据达到一定的数量的时候才持久化成上面的形式到文件系统里。

随着数据的变化，Cell Store 在逐渐地增加，这样就会减慢查询的效率。为了解决这个问题，需要后台有进程合并已经持久化的Cell Store。这一个过程叫做merging compaction（big table术语）。这也是丢弃无用记录的最恰当时机。这个过程就类似一个归并排序的过程，如图 2.22 所示。

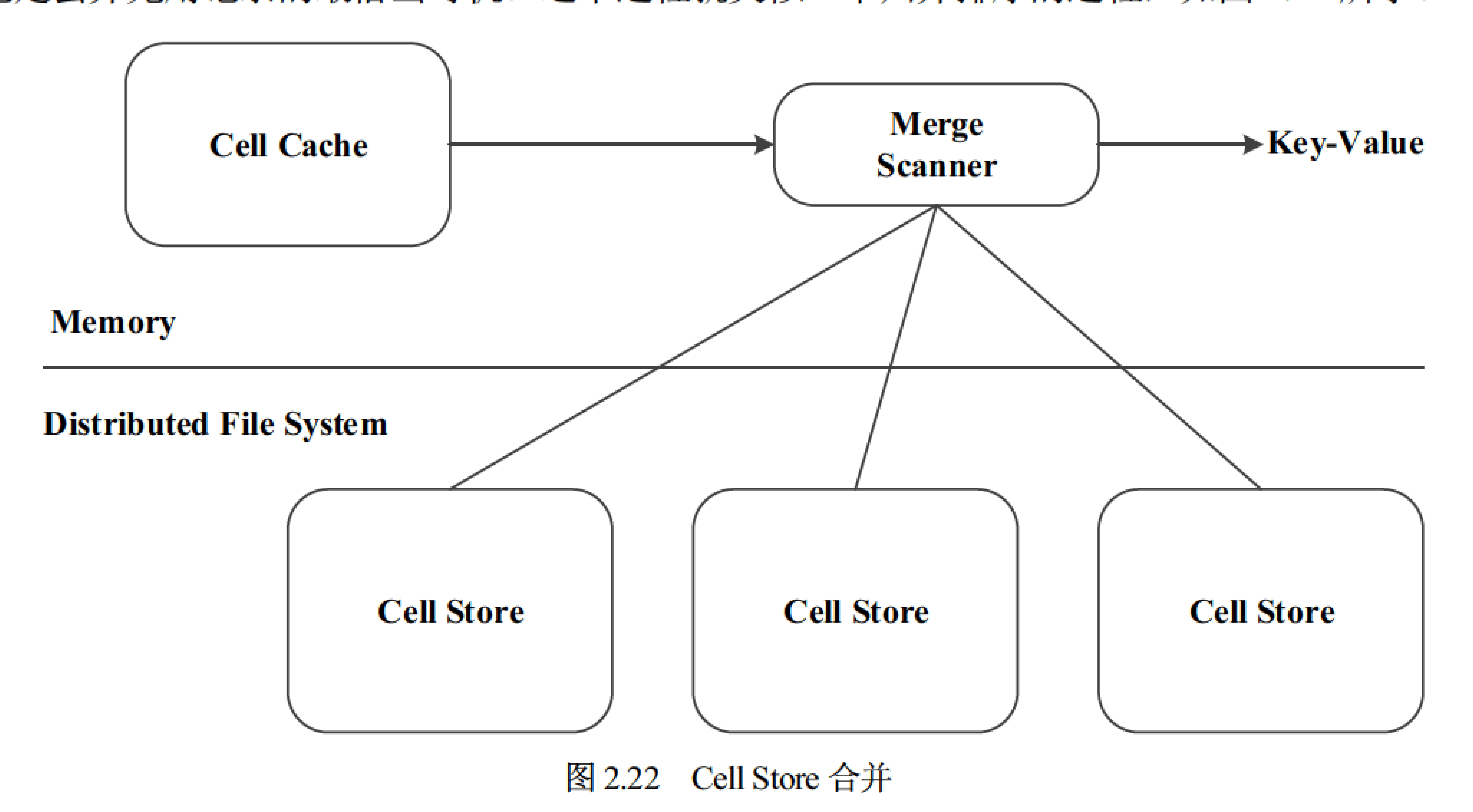


图 2.22　Cell Store 合并

2.4.8　其他开源 NoSQL 数据库

Neo4j

Neo4j（www.neo4j.org/）是一个嵌入式，基于磁盘的、支持完整事务的 Java 持久化引擎，它在

图像中而不是表中存储数据。Neo4j 提供了大规模可扩展性，在一台机器上可以处理数十亿节点/关系/属性的图像，可以扩展到多台机器并行运行。相对于关系数据库来说，图形数据库善于处理大量复杂、互连接、低结构化的数据，这些数据变化迅速，需要频繁查询——在关系数据库中，这些查询会导致大量的表连接，因此会产生性能上的问题。Neo4j 重点解决了拥有大量连接的传统 RDBMS 在查询时出现的性能衰退问题。通过围绕图形进行数据建模，Neo4j 会以相同的速度遍历节点与边，其遍历速度与构成图形的数据量没有任何关系。 此外，Neo4j 还提供了非常快的图形算法、推荐系统和 OLAP 风格的分析，而这一切在目前的 RDBMS 系统中都是无法实现的。

Riak

Riak（basho.com/riak/）是一款非常适合于 Web 应用程序的数据库，它提供了去中心化的 Key/Value 存储、灵活的 map/reduce 引擎和友好的 HTTP/JSON 查询接口。它是一个真正的容错系统，不会出现单点故障，在 Riak 世界中，没有哪台机器是特殊的或属核心服务器，它们都是对等的。

Oracle Berkeley DB

Oracle Berkeley DB（http://www.oracle.com/technetwork/cn/products/berkeleydb/overview/index.html）是一系列开源的嵌入式数据库，使开发人员能够将一个快速、可伸缩、具有工业级别的可靠性和可用性的事务处理数据库引擎结合进他们的应用程序中。 Berkeley DB 最先是由伯克利加州大学为了移除受到 AT&T 限制的程式码，从 BSD 4.3 到 4.4 时所改写的软件。Berkeley DB 运行在大多数的操作系统中，例如大多数的 UNIX 系统、Windows 系统，以及实时操作系统。

Apache Cassandra

Cassandra（cassandra.apache.org/）是一款高可扩展性第二代分布式数据库，属于混合型的非关系的数据库，类似于 Google 的 BigTable，支持的数据结构非常松散，类似于 JSON 的 BJSON 格式，因此可以存储比较复杂的数据类型。Cassandra 最初由 Facebook 开发，后转变成了开源项目。 Cassandra 的主要特点就是它不是一个数据库，而是由一堆数据库节点共同构成的一个分布式网络服务，对 Cassandra 的一个写操作，会被复制到其他节点上去，对 Cassandra 的读操作，也会被路由到某个节点上面去读取。对于一个 Cassandra 群集来说，扩展性能是比较简单的事情，只要在群集里面添加节点就可以了。Facebook、Digg、Twitter 和 Cisco 等大型网站都使用了 Cassandra。

Memcached

Memcached（www.memcached.org/）是开源的分布式 cache 系统，现在很多的大型 Web 应用程序包括 facebook、youtube、wikipedia、yahoo 等都在使用 memcached 来支持每天数亿级的页面访问。通过把 cache 层与它们的 Web 架构集成，应用程序在提高了性能的同时，还大大降低了数据库的负载。

Memcached 处理的原子是每一个 key/value 对，key 会通过一个 hash 算法转化成 hash-key，便于查找、对比以及做到尽可能的散列。同时，memcached 使用的是一个二级散列，通过一张大 hash 表来维护。

Keyspace

Keyspace（keyspace.net/）是一家叫做 Scalien 的创业公司开发的高可靠 key/value 存储系统，Keyspace 强调的技术点是高可靠性，有以下一些特点：

- Key/Value 存储：一个 key/value 数据存储系统，只支持一些基本操作，如 SET(key, value) 和 GET（key）等。
- 分布式：多台机器(nodes)同时存储数据和状态，彼此交换消息来保持数据一致，可视为一个完整的存储系统。为了更可靠，Keyspace 推荐使用奇数个 nodes，如 3、5、7 等。
- 数据一致：所有机器上的数据都是同步更新的，不用担心得到不一致的结果，Keyspace 使用著名的 Paxos 分布式算法。
- 冗余：所有机器(nodes)保存相同的数据，整个系统的存储能力取决于单台机器(node)的能力。
- 容错：如果有少数 nodes 出错，比如重启、当机、断网、网络丢包等各种 fault/fail 都不影响整个系统的运行。
- 高可靠性：容错、冗余等保证了 Keyspace 的可靠性。

MariaDB

MariaDB（mariadb.org/）是一个向后兼容的，旨在替换 MySQL 数据库的 MySQL 分支，它包括所有主要的开源存储引擎，另外也开发了属于自己的 Maria 存储引擎。MariaDB 是由原来 MySQL 的作者 Michael Widenius 创办的公司所开发的免费开源数据库服务器，与 MySQL 相比较，MariaDB 更强的地方在于：

- Maria 存储引擎
- PBXT 存储引擎
- XtraDB 存储引擎
- FederatedX 存储引擎
- 更快的复制查询处理
- 线程池
- 更少的警告和 bug
- 运行速度更快
- 更多的 Extensions (More index parts, new startup options etc)
- 更好的功能测试
- 数据表消除
- 慢查询日志的扩展统计
- 支持对 Unicode 的排序

Drizzle

Drizzle（www.drizzle.org/）是从 MySQL 衍生出来的一个数据库，但它的目的不是要取代 MySQL，它的宗旨是构建一个“更精练、更轻量、更快速”的 MySQL 版本，它的扩展性和易用

性与 MySQL 相当，但为了提高性能和扩展性，它从原来的核心系统里移除了部分功能。Drizzle 是一种为云和网络程序进行了特别优化的数据库，它是为在现代多 CPU/多核架构上实现大规模并发而设计的。

HyperSQL

HyperSQL（hsqldb.org/）是用 Java 编写的一款 SQL 关系数据库引擎，它的核心完全是多线程的，支持双向锁和 MVCC（多版本并发控制），几乎完整支持 ANSI-92 SQL，支持常见数据类型，最新版本增加了对 BLOB 和 CLOB 数据的支持，最高支持达 64TB 的数据量。同时，HyperSQL 也是一个不错的嵌入式数据库。

MonetDB

MonetDB（www.monetdb.org/Home）是一个高性能数据库引擎，主要用在数据挖掘、OLAP、GIS、XML Query、文本和多媒体检索等领域。MonetDB 对 DBMS 的各个层都进行创新设计，如基于垂直分片的存储层、为现代 CPU 优化的查询执行架构、自动和自助调整索引、运行时查询优化，以及模块化的软件架构。MonetDB/SQL 是 MonetDB 提供的关系数据库解决方案，MonetDB/XQuery 是 XML 数据库解决方案，MonetDB Server 是 MonetDB 的多模型数据库服务器。

Persevere

Persevere（www.persvr.org）是针对 Javascript 设计的基于 REST 的 JSON 数据库、分布式计算、持久对象映射的框架，提供独立的 Web 服务器，主要用于设计富客户端应用， 可以用在任何框架和客户端上。Persevere Server 是一个基于 Java/Rhino 的对象存储引擎，在交互式的客户端 JavaScript 环境中提供持久性的 JSON 数据格式。

eXist-db

eXist-db（exist-db.org）是使用 XML 技术构建的数据库存储引擎，它根据 XML 数据模型存储 XML 数据，提供高效的、基于索引的 XQuery 查询。eXist-db 支持许多 Web 技术标准，使得它非常适合 Web 应用程序开发：

- XQuery 1.0 / XPath 2.0 / XSLT 1.0 (使用 pache Xalan)或 XSLT 2.0
- HTTP 接口：REST，WebDAV，SOAP，XMLRPC，Atom 发布协议
- XML 数据库规范：XMLDB，Xupdate，XQuery 更新扩展
- 最新的 1.4 版本还增加了基于 Apache Lucene 的全文索引，轻量级 URL 重写和 MVC 框架以及对 XProc 的支持。eXist-db 与 XQuery 标准高度兼容(目前 XQTS 的得分是 99.4%)。

Gladius

Gladius（freecode.com/projects/gladiusdb）是用纯 PHP 编写的平面文件数据库引擎，它的 SQL 语法与 SQL92 的一个子集兼容，它捆绑了一个轻量级的 ADODB 驱动。

CloudStore

CloudStore（gcloud.civilservice.gov.uk/cloudstore/）（以前叫做 Kosmos 文件系统）是一个开源

的高性能分布式文件系统，它是用 C++编写的，CloudStore 可以和 Hadoop 以及 Hypertable 集成，这样就允许应用程序构建在那些系统上，而底层数据存储无缝地使用 CloudStore。CloudStore 支持 Linux 和 Solaris，主要用来存储 Web 日志和 Web 爬行数据。

OpenQM

OpenQM（www.openqm.org/）是唯一一款同时有商业支持和免费的开源多值数据库，基于 GPL 协议发布，多值数据库对 NoSQL 运动起到了推动作用，它自身也因速度快，体积小，比关系数据库便宜而得到了认可。名称 OpenQM 中的 Open 表示开源版本，QM 表示商业闭源 QM 数据库。商业版本支持 Windows、Linux（RedHat，Fedora，Debian，Ubuntu）、FreeBSD、Mac OS X 和 Windows Mobile，其列表价格还不到其他多值产品的 1/5，商业版本还包括一个 GUI 管理界面和终端模拟器，开源版本仅包括核心多值数据库引擎，主要是为开发人员准备的。

ScarletDME

ScarletDME（https://www.ohloh.net/p/ScarletDME）也是一个开源多值数据库，它是 OpenQM 的社区分支版，最初由 Ladybridge 开发，这个项目创立于 2008 年 11 月 28 日，它既在独立开发自己的功能，也在为 OpenQM 贡献代码。这个项目最初的名字叫做 Ladybridges GPL OpenQM，现在正式改为 ScarletDME，其中的 DME 是 Data Management Environment（数据管理环境）的首字母缩写。

SmallSQL

SmallSQL（www.smallsql.de）是一个 100%纯 Java 编写的轻量级数据库，一般用于嵌入式领域，兼容 SQL 99 标准，支持 JDBC 3.0 API，定位于高端 Java 桌面 SQL 数据库。支持所有能运行 Java 的平台，可直接嵌入到应用程序中。不过它也有一些不足，如没有网络接口、必须安装 Java 运行时、同一时间不能在多个应用程序之间共享数据库、没有用户管理。

LucidDB

LucidDB（www.luciddb.org）是唯一一款专注于数据仓库和商务智能的开源 RDBMS，它使用了列存储架构，支持位图索引，哈希连接/聚合和页面级多版本，大部分数据库最初都注重事务处理能力，而分析功能都是后来才加上去的。相反，LucidDB 中的所有组件从一开始就是为满足灵活的需求，高性能数据集成和大规模数据查询而设计的，此外，其架构设计彻底从用户出发，操作简单，完全无需 DBA。

LucidDB 对硬件要求也极低，即使不搭建集群环境，在单一的 Linux 或 Windows 服务器上也能获得极好的性能。最新版本还加入了对 Mac OS X 和 Windows 64 位的支持，官方网站上的文档和教程也非常丰富，非常值得体验一下。

HyperGraphDB

HyperGraphDB（www.hypergraphdb.org）是一种通用的、可扩展的、可移植的、分布式、嵌入式和开源数据存储机制，它是一个图形数据库，专门为人工智能和语义 Web 项目而设计，它也可用于任意规模的嵌入式面向对象的数据库。正如其名，HyperGraphDB 是用来存储超图的，但

它也属于一般图形数据库家族，作为一个图形数据库，它不施加任何限制，相比其他图形数据库它的功能更丰富。

HyperGraphDB 非常稳定，已经应用在多个生产环境，包括一个搜索引擎和 Seco scripting IDE。它支持*nix 和 Windows 平台，需要 Java 5+。

InfoGrid

InfoGrid（infogrid.org/）是一个互联网图形数据库，它提供了许多额外的组件，使得在图像基础上开发 RESTful Web 应用程序变得更加容易。InfoGrid 是开源的，包括一系列项目：

- InfoGrid 图形数据库项目——InfoGrid 的心脏 GraphDatabase，可以独立使用，也可以附加到其他 InfoGrid 项目。
- InfoGrid 图形数据库网格项目——在 GraphDatabase 基础上增加了复制协议，因此多个分布式 GraphDatabase 就可以在一个非常大的图像管理环境中协作。
- InfoGrid 存储项目——像 SQL 数据库和分布式 NoSQL 哈希表那样，为存储技术提供一个抽象的通用接口，这样 InfoGrid GraphDatabase 就可以使用任何存储技术持久化数据。
- InfoGrid 用户接口项目——将 GraphDatabase 中的内容以 REST 风格映射成浏览器可访问的 URL。
- InfoGrid 轻量级身份识别项目——实现以用户为中心的身份识别技术，如 LID 和 OpenID。
- InfoGrid 模型库项目——定义一个可复用对象模型库，作为 InfoGrid 应用程序的模式使用。
- InfoGrid Probe 项目——实现 Probe 框架，它允许开发人员将任何互联网上的数据源当作一个图像对象看待。
- InfoGrid Utilities 项目——收集 InfoGrid 使用的常见对象框架和实用代码。

Apache Derby

Apache Derby（http://db.apache.org/derby/）是 Apache DB 的子项目，它完全用 Java 编写，是一个开源关系数据库，它的体积非常小，基础引擎加上 JDBC 驱动只有 2.6MB，它支持 SQL 标准，提供了一个嵌入式 JDBC 驱动，因此可以嵌入到任何基于 Java 的应用程序中，Derby 也支持常见的客户端/服务器模式，它也易于安装和使用。

Hamsterdb

Hamsterdb（hamsterdb.com/）是一个轻量级嵌入式 NoSQL Key/Value 存储引擎，它已经有 5 年历史，现在它的开发重点放在易用性、高性能、稳定性和可扩展性上。Hamsterdb 支持事务(同一时间只能处理一个事务)，支持内存数据库。支持基于 HTTP 服务器的嵌入式远程数据库，支持日志/恢复、AES 加密、基于 zlib 的压缩、支持 C++、Python、.NET 和 Java 编程语言。

H2 Database

H2 Database（www.h2database.com/）是一个开源的 Java 数据库，它的速度很快，包括 JDBC API，支持嵌入式和服务器模式，内存数据库，提供了一个基于浏览器的控制台程序，它的体积也非常小，只有一个大小约 1MB 的 jar 文件，它还支持 ODBC 驱动和全文搜索。

EyeDB

EyeDB（www.eyedb.org/）是一款基于 ODMG 3 规范的面向对象数据库管理系统，为 C++和 Java 提供了编程接口，它功能非常强大，并且成熟、稳定和安全，实际上，它起源于 1992 年的 Genome View 项目，1994 年又进行了重写，广泛用于生物信息项目。

txtSQL

txtSQL（http://sourceforge.net/projects/txtsql/）是一个面向对象的平面文件数据库管理系统，它使用 PHP 编写，支持对普通文本文件的操作，虽然是一个文本数据库，但同样支持 SQL 的一个子集，并且执行效率非常高，txtSQL 使用文件系统的方法与 MySQL 的表和数据库原理类似，它有一个类似于 phpMyAdmin 的管理界面。

db4o

db4o（www.db4o.com/）是一个面向对象的开源数据库，允许 Java 和.NET 开发人员用一行代码存储和检索任何应用程序对象，无需预定义或维护一个独立的、僵化的数据模型，因为模型现在是由 db4o 根据需要自动创建和更新的。db4o 成功的秘密是因为它的易用性，它原生为 Java 和.NET 设计，存储数据对象的方法直 接在应用程序中定义，因此 db4o 很容易集成到应用程序中，由于只需要一行代码，因此执行效率非常高。

Tokyo Cabinet

Tokyo Cabinet（tokyocabinet.sourceforge.net）是一个 Kay/Value 型数据库，每个 Key 和 Value 的长度都可以不同，Kay 和 Value 既可以是二进制数据，也可以是字符串，无数据表和数据类型的概念，记录是以哈希表、B+树和固定长度数组形式组织的。Tokyo Cabinet 具有以下优点：

- 空间利用率高——数据文件尺寸更小。
- 执行效率高——更快的处理速度。
- 并发性能好——在多线程环境性能更好。
- 改善的可用性——简化的 API。
- 改善的可靠性——即使在发生灾难的情况下，数据文件也不会损坏。
- 支持 64 位架构——支持海量的存储空间和巨型数据库文件。

Tokyo Cabinet 是用 C 语言编写的，为 C、Perl、Ruby、Java 和 Lua 提供了 API。

Voldemort

Voldemort（www.project-voldemort.com/）是一个分布式 Key/Value 存储系统，它具有以下特点：

- 数据自动在多个服务器之间复制。
- 数据自动分区，因此每个服务器只包括整体数据的一个子集。
- 服务器故障处理是透明的。
- 支持插入式序列化，允许丰富的 Key 和 Value 类型，包括列表和元组，也可以集成常见的序列化框架，如 Protocol Buffers、Thrift、Avro 和 Java Serialization。
- 数据项支持版本化，即使在故障情况下，数据完整性也可以得到保障。

- 每个节点都是独立的，无需其他节点协调，因此也没有中央节点。
- 单节点性能优秀：根据机器配置、网络、磁盘系统和数据复制因素的不同，每秒可以执行 10~20k 操作。
- 支持地理分散式部署。

2.5 本章小结

本章介绍了数据存储技术的概念与发展过程，讨论了海量数据存储的一些关键性问题，并介绍了一些具有典型代表意义的 NoSQL 数据库产品，表 2.1 对这些产品的特征进行了总结。

表 2.1　NoSQL 数据库总结

类型	NoSQL 数据库代表	特点
列存储	HBase Cassandra Hypertable BigTable	按列存储数据的，方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的 IO 优势，具有高可扩展性、可用性、面向分布式计算的特点
文档型存储	MongoDB CouchDB	文档存储一般用类似 json 的格式存储，存储的内容是文档型的，这样也就有机会对某些字段建立索引，实现关系数据库的某些功能，满足海量存储和访问的数据库
Key-value 存储	Tokyo Cabinet/Tyrant Berkeley DB MemcacheDB Riak Keyspace Voldemort	可以通过 key 快速查询到其 value。一般来说，存储不管 value 的格式，照单全收，满足极高读写要求
图存储	Redis Neo4J FlockDB	图形关系的最佳存储，使用传统关系数据库来解决的话性能低下，而且设计使用不方便
对象存储	Db4o Versant txtSQL EyeDB	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据
XML 数据库	Berkeley DB XML BaseX	高效的存储 XML 数据，并支持 XML 的内部查询语法，比如 XQuery、Xpath

第 3 章

数据抽取和清洗

大数据的一个重要特点就是多样性，这就意味着数据来源极其广泛，数据类型极为繁杂。这种复杂的数据环境给大数据的处理带来极大的挑战。要想处理大数据，首先必须对所需数据源的数据进行抽取和集成，从中提取出关系和实体，经过关联和聚合之后采用统一定义的结构来存储这些数据。在数据集成和提取时需要对数据进行清洗，保证数据质量及可信性。本章就介绍了各种数据抽取和清洗技术及其实现。

数据抽取作为数据处理的第一步，具有至关重要的作用。大数据量对应着海量噪杂的信息，不可避免地带来大数据困惑。如何从大数据中提取关键性的代表性特征，可能是某些词汇，也可能是某些短语、命名实体或流行用语，则成为大数据分析的一把利器。

随着网络和信息技术的发展，出现了“信息爆炸”的问题，即数据极其丰富而所需知识相对匮乏。人们所需求的数据分散在多家网站的 Web 网页上，为了得到自己所需的信息，不得不在浩如烟海的网页中搜索、浏览，寻找符合自己所需的知识，不仅浪费了大量的时间和精力，而且有时不一定能得到自己所需的知识，所以说在数据极大丰富的同时，也带来了数据泛滥的问题，Web 数据转换集成技术正是用来从巨量的信息中获取有效信息的方法。如何快速、准确地从海量数据里面提取有用的信息已经成为当前计算机科学的关注热点。

Web 数据集成技术可以从 Web 上自动获取数据，然后集成为用户所关心的有效信息，并在此基础上实现高效的查询、检索和比较，乃至数据挖掘、知识发现等应用。但是由于 Web 数据的特点，从 Web 上得到的数据中有可能存在着大量的脏数据，引起的主要原因有：滥用缩写词、惯用语、数据输入错误、重复记录、丢失值、拼写变化、不同的计量单位等。如果其中存在着大量的脏数据，那么这些数据也是没有任何意义的，根本就不可能为以后数据挖掘决策分析系统提供任何支持。没有数据清洗，很可能就会导致错误的决策，因此数据清洗是构建数据仓库和知识发现的必要因素。

如何有效地保证数据质量是关系到信息抽取和数据挖掘是否成功的问题，对此问题解决方案的探讨已经经成为当今软件技术的一个新的研究热点。本章的研究内容就是在上述背景下提出的，采取的手段是利用数据清洗技术来消除各种脏数据，从而实现保证数据质量的目标。

3.1 数据抽取和清洗技术介绍

3.1.1 数据抽取简介

百度给数据抽取下了一个定义：数据抽取是从数据源中抽取数据的过程。数据源采用关系型数据库和非关系数据库。具体来说，就是搜索整个数据源，使用某些标准选择合乎要求的数据，并把这些数据传送到目的文件中。简单来说，数据抽取就是从数据源中抽取数据的过程。数据源可以简单分为结构化数据、半结构化数据和非结构化数据。

数据的抽取需要在调研阶段做大量工作，首先要搞清楚数据是从很多业务系统中来的，每个业务可能都有各自的数据库，是否有非结构化数据等。大数据与传统海量数据的差别主要在于海量数据一般都是指存储在数据库中的结构化数据，而大数据面对的则是大量非结构化的业务数据，如招标公告文本、采购文本中的各类有价值的项目数据、招标金额、产品规格信息。下面简单介绍几种不同数据源的处理方法。

1. 与存放 DW 的数据库系统相同的数据源处理方法

这一类数据源在设计时比较容易，一般情况下，DBMS（包括 SQLServer，Oracle）都会提供数据库链接功能，在 DW 数据库服务器和原业务系统之间建立直接的链接关系就可以写 Select 语句直接访问。

2. 与 DW 数据库系统不同的数据源的处理方法

这一类数据源一般情况下也可以通过 ODBC 的方式建立数据库链接，如 SQL Server 和 Oracle 之间。如果不能建立数据库链接，可以有两种方式完成，一种是通过工具将源数据导出成.txt 或者是.xls 文件，然后再将这些源系统文件导入到 ODS 中。另外一种方法通过程序接口来完成。

3. 文件类型数据源（.txt，.xls）

可以培训业务人员利用数据库工具将这些数据导入到指定的数据库，然后从指定的数据库抽取。或者可以借助工具实现，如 SQL Server 2005 的 SSIS 服务的平面数据源和平面目标等组件导入 ODS 中去。

4. 增量更新问题

对于数据量大的系统，必须考虑增量抽取。一般情况，业务系统会记录业务发生的时间，可以用作增量的标志，每次抽取之前首先判断 ODS 中记录最大的时间，然后根据这个时间去业务系统取大于这个时间的所有记录。利用业务系统的时间戳，一般情况下，业务系统没有或者部分有时间戳。

在信息系统设计中，我们会将处理的数据按业务模型进行分类。通常用实体（Entity）及关系

（Relation）来描述数据，构成关系数据模型。在面向对象的设计中，我们按照对象本身的特性来设计数据结构。这些数据都具有固定的字段语义、明确的数据类型，具有数据值范围及可施加的操作限制等约束规范。这些数据称为结构化数据（Structured Data）。

而声音、图片、视频或一段描述性文本等数据，在信息系统设计中一般不直接描述其内部结构，而是作为一个整体来处理，这种数据称为非结构化数据（Unstructured Data ）。

所谓半结构化数据（Semi-structured Data ），就是介于结构化数据和非结构化数据之间的数据形式，HTML 文档就属于半结构化数据。Web 页面一般由后台数据库驱动，但这些数据源中的结构化数据无法直接获取，只能透过 HTML 这样的视图来获得半结构化数据。

数据抽取问题可以用下面的定义：给定数据源 S，确定一个 S 到数据库的映射 M，该映射从 S 中抽取数据对象并将这些数据对象按一定的格式组装（Populate）到 R 中。实现这一映射的计算机程序就是数据抽取程序，俗称包装器（Wrapper）。要构造一个可以正常工作的包装器，必须回答下列问题。

（1）要在数据源中抽取怎样的数据对象

数据源中的数据对象可以是简单的字符串，也可以具有树形结构，甚至具有有向图结构（其中结点又称数据类型）。为使包装器具有通用性，通常用数据抽取模型来描述数据源中数据对象的结构。

（2）如何在数据源中找到这些数据对象

通常用抽取规则驱动一个通用抽取算法在数据源中搜索与抽取规则匹配的数据对象。

（3）用什么格式组装找到的数据对象

通常用符合某个数据库模式的格式来组装找到的数据对象，这个数据库模式可以是关系的、对象关系的、面向对象的或是 XML-Schema 的。

（4）如何将找到的数据对象组装到数据库中

通常的方法是用一组映射规则描述数据类型到数据库字段之间的对应关系。当抽取算法找到一个数据对象时，先用映射规则根据数据对象所属的数据类型找到对应的数据库字段，然后将这个数据对象组装到这个字段中。一般并不单独设计组装算法，数据对象的组装通常在抽取算法中进行。因此抽取算法有时也称为抽取和组装算法。

（5）如何生成和维护数据抽取过程所需的元数据

为了使数据抽取和组装算法正常工作，必须向它提供数据抽取模型、抽取规则、数据库模式和映射规则等参数，本书将这些参数称为元数据。将数据仓库系统中的元数据定义为“用于支持数据仓库管理和有效应用的任何信息”（本章的元数据和这一定义类似，不同之处是这里仅涉及数据抽取问题）。一个数据源需要用一套元数据进行描述。由于数据集成系统往往会涉及大量的数据源和相关元数据，使得生成和维护这些元数据的工作成了沉重的负担。

3.1.2　数据清洗简介

数据清洗原理即通过分析“脏数据”的产生原因和存在形式，利用现有的技术手段和方法去清洗“脏数据”，将原有的不符合要求的数据转化为满足数据质量或应用要求的数据，从而提高数据集的数据质量。

数据清洗的定义在不同的应用领域不完全相同。例如，在数据仓库环境下，数据清洗是抽取转换装载过程的一个重要部分，要考虑数据仓库的集成性与面向主题的需要（包括数据的清洗及结构转换）。数据清洗主要是提高数据的可利用性（去除噪声、无关数据、空白数据域，考虑时间顺序和数据的变化等），但主要内容还是一样的。数据清洗是一个减少错误和不一致性、解决对象识别的过程。可以这么定义：对数据源进行详细分析后，利用相关技术（如预定义的清洗规则、字典函数库及重复记录匹配等）将从单个或者多个数据源中抽取的脏数据经过一系列转化使其成为满足数据质量要求的数据，这样的过程称为数据清洗。

对于数据清洗的定义目前还没有共识。对于应用于不同领域中的数据清洗有不同的解释。目前，数据清洗主要应用于三个领域：数据仓库（DW）、数据库中的知识发现（KDD）和数据质量管理（TDQM）。

在数据仓库领域中，数据清洗一般是应用在几个数据库合并时或多个数据源进行集成时。指代同一个实体的记录，在合并后的数据库中就会出现重复的记录。数据清洗过程就是要把这些重复的记录识别出来并消除它们，也就是所说的合并/清洗（merge/purge）问题。这个问题的实例字面可称作记录链接、语义集成、实例识别或对象标识问题。

从这个方面讲，数据清洗可有几种定义：数据清洗是消除数据的错误和不一致并解决对象标识问题的过程；数据清洗是归并/清洗问题；数据清洗并不是简单地用优质数据更新记录，它还涉及数据的分解与重组。

TDQM 是一个学术界和商业界都感兴趣的领域。有很多论文提到数据质量及其集成问题，但是很少涉及数据清洗问题。有些文章从数据质量的角度论及过程管理问题。在数据生命周期中，数据的获取和使用周期包括系列活动：评估、分析、调整、丢弃数据。如果将数据清洗过程和数据生命周期循环集成起来，这一系列步骤就从数据质量的角度给数据清洗下了定义。

到目前为止，数据清洗还没有统一的定义，它在数据仓库、数据/信息质量管理和数据库中的知识发现（KDD）等应用领域中有不同的表述方式，下面分别在这三个应用领域中介绍数据清洗的定义。

1. 数据仓库应用中的数据清洗定义

在数据仓库领域中，同一个实体的记录在不同的数据源中以不同的格式表示或被错误地表示，那在几个数据库合并后或多个数据源集成后数据库仓库中就会存在重复记录，数据清洗就是要把这些重复的记录识别出来并消除它们，也就是所说的合并清除（merge/purge）问题，这样的过程一般简述为：记录连接、语义整合、实例识别或对象识别问题。

2. 数据/信息质量管理应用中的数据清洗定义

全面数据质量管理解决整个信息业务过程中的数据质量及其集成问题，在该领域中，没有直接定义数据清洗，应用于数据获取和数据使用的数据质量生命周期模型（包括数据的评价、分析、调整和丢弃等），是从数据质量的角度，把数据清洗过程和数据生命周期集成在一起，因此，数据清洗过程被定义为一个评价数据正确性并改善其质量的过程。

3. KDD 应用中的数据清洗定义

在 KDD 领域，数据清洗被认为是 KDD 过程的一个步骤，即对数据进行预处理的过程。各种不同的 KDD 和 DM 系统都是针对特定的应用领域进行数据清洗，数据清洗是一种使用计算机化的方法来检查数据库，检测丢失的和不正确的数据，并纠正错误数据的过程。

在 KDD 的过程中，数据库可能包含一些数据对象，它们与数据的一般行为或模型不一致，这些数据对象被称为异常记录。异常记录分成两部分。

- 一部分的记录可能反映某种稀有而真实的情况，比如保险欺诈、违规交易、信用卡欺诈等，罕见的事件可能比正常出现的事件更有意义。
- 另一部分的记录是在产生数据样本的过程中受干扰或污染的结果，比如拼写问题、打字错误、不合法值、空值、不一致值、简写、同一实体的多种表示（重复）等，这部份记录也是数据清洗中的“脏数据”，根据“垃圾进、垃圾出”的原理，在大部分应用系统中，比如数据仓库、KDD 及综合数据质量管理等，必须先对脏数据进行清洗。这样一方面能提高数据质量，另一方面能增强数据的可用性，检测异常记录中的脏数据也是本书研究的重要目标，即设计加权快速聚类算法，以便高效快速检测异常记录，设计基于等级分组的相似重复记录检测算法，以便高效快速地检测重复记录。

数据清理定义为发现和清除数据中的错误和不一致来提高数据的质量，那么数据清洗的任务就是通过各种措施从准确性、一致性、无冗余、符合应用的需求等方面提高已有数据的质量。大多数数据质量问题可以在数据录入的初期阶段发现并解决，称之为预防阶段。处理目标通常情况只是单一记录，因此只要具有充分的业务规则就能基本保障录入数据的质量，然而我们面临的一个更为迫切的问题是如何充分利用已经存在的大量历史数据和各种原始信息源中的数据。数据采集和数据库技术的快速发展产生了大量的历史数据库，这些数据库的数据往往由同样年代久远的信息系统产生和维护。由于这些信息系统本身的设计和当时各种技术的局限性，使得这些历史数据库中的数据质量不能满足现在的多种应用程序的要求，因此就必须采取一定的措施处理这些数据，使之符合新的应用场景。

海量数据的应用正是为了满足这样的需求。而建立数据仓库的过程中往往需要集成历史数据、现行数据以及来自不同数据源的各种数据。据统计，在建立数据仓库的过程中 75%的工作量将投入到类似数据准备和数据装载这样的后端事务中。数据仓库中数据质量的好坏是数据仓库应用成功与否的关键因素。许多数据仓库项目的失败就是因为对导入数据仓库的各种数据质量缺乏足够的重视。因此建立数据仓库的过程中，需要有效的技术手段和工具来提高导入数据仓库的数据质量。

数据清洗是一个非常复杂的任务，并且包含着一些互相关联的问题。一方面转换必须尽可能具有通用性，而且不依赖大量的编程工作，也就是支持在多领域内的多种错误检测算法；另一方面，系统需要支持一种简单的接口定义来进行错误检测和数据转换。因此，需要在现有的算法和功能基础上，设计相应的数据清洗系统，使它具有相对的通用性和可交互性。通用性的主要实现技术有数据标准化、术语化、通用的接口标准，即通用的过程描述语言，通用的数据结构以支持用户进行扩展。可交互性支持用户通过系统反馈的检测统计图表，实时地修改转换过程，避免用户与系统的隔离。

数据清洗按照实现方式与范围，可分为 4 种。

（1）手工实现

通过人工检查，只要投入足够的人力物力财力，也能发现所有错误，但效率低下。在大数据量的情况下，几乎是不可能的。

（2）编写专门的应用程序

这种方法能解决某个特定的问题，但不够灵活，特别是在清洗过程需要反复进行（一般来说，数据清洗一遍就达到要求的很少）时导致程序复杂，清洗过程变化时工作量增大。

（3）解决某类特定应用领域的问题

如根据概率统计学原理查找数值异常的记录，对姓名、地址、邮政编码等进行清洗，这是目前研究较多的一类问题，也是应用比较成功的一类数据清洗问题。

（4）与特定应用领域无关的数据清洗

这一部分的研究主要集中在清洗重复的元素上。

在这 4 种实现方法中，由于后两种具有某种通用性及其较大的实用性，已受到了越来越多的关注。但是不管哪种方法，都由数据分析/定义、搜索/识别错误元素、修正错误 3 个阶段组成。

第一阶段：尽管已有一些数据分析工具，但仍以人工分析为主。可以将错误类型分为两大类，即单数据源与多数据源，并将它们又各分为结构级与记录级错误。这种分类非常适合于解决数据库中的数据清洗问题。

第二阶段：有两种基本的思路用于识别错误。一种是发掘数据中存在的模式，然后利用这些模式清洗数据；另一种是基于数据的，根据预定义的清洗规则，查找不匹配的记录。后者用得较多。

第三阶段：某些特定领域能够根据发现的错误模式，编制程序或借助于外部标准源文件、数据字典等在一定程度上对错误进行修正，对数值字段进行更新。有时能根据数理统计知识自动修正，但经常须编制复杂的程序或借助于人工干预完成。

3.2 数据抽取和清洗研究现状

信息抽取研究从 1980 年末开始蓬勃开展，消息理解会议（Message Understanding Conference，简称 MUC）起到了重要的推进作用。MUC 系列会议的召开使信息抽取技术逐步发展成为自然语言处理领域的一个重要分支，且持续推动这个研究领域的发展。

MUC 会议，由美国国防高级研究计划委员会资助。从 1987 年到 1998 年共举行了七届。每次会议之前，组织方先向各参会者安排信息抽取任务，并提供消息文本的样例和抽取任务的说明，让参会者开发相关信息抽取系统，要求能够处理这种消息文本。在正式会议开始前运行各自的系统，对给定的消息文本集进行处理和测试。根据各系统的输出结果与标准结果的对照情况，得到最终的评测结果。然后才开始正式的会议，会议时由参与者相互交流体会和想法。这种基于评测驱动的会议模式也得到了广泛推广。

1993 年 8 月举行的第 5 次 MUC 会议有一个重要创新，引入了嵌套的模板结构。通过借鉴面向对象的思想和框架知识表示的思想，信息抽取模板由多个子模板组成，而不再是单个模板的扁平结构。MUC 系列会议的召开使信息抽取这一新的研究方向的确立和发展起到了重大的推动作用。MUC 定义的信息抽取任务的各种规范以及相应的评价体系，已经成为信息抽取。近年来，信息抽取技术的研究与应用更为活跃。在研究方面，主要侧重于以下几方面：利用机器学习技术增强系统的可移植能力、探索深层理解技术、篇章分析技术、多语言文本处理能力、WEB 信息抽取以及对时间信息的处理等。

迄今为止，国内的研究基本上都处在包装器的半自动生成阶段，能自动识别网页并完全自动生成包装器的数据抽取方法的研究尚未见诸文献。目前，国内较为典型的数据抽取算法和抽取系统主要有以下几种。

- 中国人民大学提出的基于预定义模式的包装器。采用的方法是由用户定义模式并给出模式与 HTML 页面之间的映射关系，然后由系统推导出抽取规则，同时生成包装器。
- 中科院软件所提出的基于 DOM 的信息提取算法。该算法以文档对象模型（Document Object Model，DOM）为基础，把所要提取的信息位于 DOM 层次结构中的路径作为信息抽取的“坐标”。在这个基本原理的基础上，设计了一种半自动生成抽取规则的归纳学习算法，然后根据抽取规则自动生成 Java 类，并将该类作为 Web 数据抽取包装器组成的重要构件。
- 河北大学提出的基于样本实例的 Web 信息抽取。用户首先指定样本页面和预先定义模式的对象关系模型。然后对样本页面和其中的样本记录进行标记学习，形成规则（包含抽取规则和关联规则），并将规则放入知识库中。最后利用知识库对其他同类页面自动抽取信息，抽取的结果存放到对象关系数据库中。
- 中科大提出的基于多层模式的多记录网页信息抽取方法。基本思想是把 HTML 网页中的待抽取信息用多层模式进行描述，利用各层模式之间相互联系的特点，动态获取各层中与 HTML 页面内容的具体描述（格式）密切相关的信息识别模式知识。

- 北京大学提出的 DAE（DOM Based Automatic Extraction）算法。其核心是借鉴 Ontology 中的 ALIGN 算法的设计思想，并对其进行了改进。DAE 算法利用 HTML 的 DOM 树的特性解决可选项的问题。其基本思想是比较不同页面间的相似和不同之处，得出一个公共的包装器，然后对该包装器进行语义分析。产生数据模式后，利用包装器将数据抽取出来并连同数据模式存放到 XML 数据集中。DAE 系统和早先的技术相比，在自动化程度上有了很大提高。
- 大连海事大学提出的 GALR 算法。是对 WHISK 算法的改进，基本思想是把遗传算法引入到抽取规则的学习中，即在扩展规则时随机增加项，以较优的目标函数指导规则扩展，以得到准确率较高的规则。这种算法比原先的技术，在准确性上获得了提高。

然而这些算法都有一定的局限性。首先，需要有较多的人工干预和指导。因为需要较多的先验知识和基础条件，且不同的系统使用的描述语言和表达形式不同，所以不仅要求进行干预的专业人员需要对网页的结构分析和生成技术等方面较为熟悉，并且还要求对系统使用的描述语言和表达形式有相当了解，才能应用到具体项目中，因此对人员的要求比较严格。

其次，根据一定的先验知识产生包装器的方法造成了系统的适应性较差，也就是说，根据某一特定情况产生的包装器只能适用于此种特定情况。当网页结构发生变化时，需要重新进行人工干预和标识，因此难以较好地适应变化，系统的可维护性较差。

针对数据质量的现状，很多学者提出了数据清洗的框架。但是数据清洗是一个领域相关性非常强的工作，而且数据质量问题非常零散、复杂、不一致，到目前为止没有形成通用的国际标准，只能根据不同的领域制定不同的清洗算法。目前的清洗算法的优良性衡量标准有以下几个方面：返回率（Recall）（重复数据被正确识别的百分率）；False-positive Error（错误地作为重复数据记录的百分比）；精确度（Precision）（算法识别的重复记录中的正确的重复记录的百分比）；计算公式（Precision=100%-False-Positive Error）。

数据清洗主要分为检测和清洗两个步骤。国内外的相关研究主要有以下几个方面：

- 提出高效的数据异常检测算法，来避免扫描整个庞大的数据集。
- 在自动检测数据异常和进行清洗处理的步骤间增加人工判断处理，以防止对正确数据的错误处理。
- 数据清洗时对数据集文件的处理。
- 如何消除合并后数据集中的重复数据
- 建立一个通用的领域无关的数据清洗框架。
- 关于模式集成的问题。

目前已经提出一些数据清洗的模型，如：基于粗糙集理论数据清洗、基于聚类模式数据清洗、基于模糊匹配数据清洗模型、基于遗传神经网络数据清洗、基于专家系统体系结构等数据清洗模型等。但这些模型都有着不同程度的弊端。以基于聚类模式的数据清洗为例：

- 已有的聚类算法对直接检测异常作用不大。在对整个记录空间运用聚类算法时能发现异常时所产生的聚类能作为下述两种方法的检测空间。其主要缺陷是耗时，特别在记录条数多

时，不适于检测异常。

- 在运用聚类算法的基础上，使用给予模式的方法（每个字段使用欧式距离，采用 K-mean 算法，K=6），仅检测到了少量记录（0.3%）满足超过 90%字段的模式。
- 经典的（布尔型）关联规则难以发现异常，但数量型关联规则、率规则（Ratio rules）、序数关联规则能较好地检测异常与错误。

目前国内对于数据清洗技术的研究，还处初级阶段。数据清洗问题的重要性是不言而喻的，然而，目前在学术界，它并没有得到足够的关注。有些人认为数据清洗是一个需要大量劳动力的过程，而且往往过于依赖特定应用领域，其实不然，在数据清洗系统的灵活框架上仍然有很多东西值得研究，在异构数据集成中，如何准确地识别相似重复记录，也有很多工作可以做。当前 Web 数据量迅速增长，对 Web 搜索引擎返回的结果进行清洗也是一个有价值的问题。随着 XML 数据处理标准的日渐成熟，如何定义 XML 文档的质量标准以及如何针对 XML 文档的数据清洗，都是值得研究的。

目前，从国内外关于数据清洗的研究现状来看，不足主要体现在以下几个方面：

- 目前数据清洗较多的是针对特定的领域。如银行、保险和证券等对客户数据的准确性要求很高的行业，都在做自己的客户数据的清洗工作，针对自己的具体应用开发软件，通用的数据清洗框架并不多见。
- 国产的数据清洗工具较少，对于不完整数据、异常数据的清洗研究较少。
- 检测重复记录的算法受到很大的关注。但是在数据量比较大时，检测效率和精度不高，耗时多，有待于更好的检测算法。
- 数据清洗的研究主要集中在数据仓库上，许多公司推出了比较成熟的 ETL 工具，但是针对 Web 数据的清洗工具很少，特别是由于 Web 数据的特点和 XML 自身所具有的特性，如通用性、自描述性。
- 由于中文与英文不同特点，数据清洗方法有所不同，针对中文的数据清洗没有引起重视。

3.3 数据抽取技术的实现

3.3.1 Web 数据抽取

1. Web 数据抽取的意义

传统的信息抽取（Information Extraction，IE）是从自然语言中获取特定的事件、事实等信息，并将这些信息以结构化的形式存入数据库中，供用户查询及进一步分析利用的过程。即它从自然语言中抽取用户感兴趣的事件、实体和关系，然后存入数据库，分析趋势，给出文摘，或进行在线服务。信息抽取还可以看作是信息检索的进一步深化，研究指定信息的查找、理解和抽取，并

将指定信息以适当的方式输出。信息抽取系统以原始文本作为输入，以固定格式的信息作为输出。

信息抽取系统的设计主要分为知识工程方法和自动训练方法。知识工程方法主要是通过手工编写规则使系统实现对特定知识领域的信息抽取。此方法要求知识工程师对知识领域有较为深入的了解；自动训练方法是给特定的语料库标记好要想得到的信息，然后通过机器学习来获取规则，这种方法比知识工程更加智能和快速，但需要足够多的训练数据作为处理质量的保证。典型的信息抽取系统由文本分块、预处理、过滤、预分析、分析、片段组合、语义解释、词汇消歧、共指消解或篇章处理、模板生成这 10 个依次相连的模块组成。

随着互联网技术的飞速发展，Web 上的网站和网页数量以爆炸性的趋势增长，从而使 Web 成为一个巨大的、分布广泛的数据源。有效地获取和集成 Web 数据，为进一步的分析和挖掘提供支持，具有十分重要的应用价值和现实意义。Web 数据集成可以实现对 Web 数据的有效整合，为市场情报分析等应用提供支持。但是，由于 Web 上的数据呈现出海量、异构、动态变化和联系丰富等特点，导致 Web 上的数据无法一次获取，一次理解，一次集成。因此，识别 Web 数据集成是一个逐渐理解 Web 数据的过程；是一个去伪存真，实现逐步净化、完善信息的过程。

Web 信息抽取是随着网络技术的发展、网页信息的扩充而产生的。它是信息抽取近年来的一个侧重分支研究领域。Web 信息抽取技术的研究和实践经历了手动到半自动再到完全自动的过程。20 世纪初，Web 信息抽取技术还处于一个半自动的发展状态。对 Web 页面的抽取通常是在程序员研究网站后，手工编写代码开发一个包装器程序，把网页的用户感兴趣的内容抽取出来存放在数据库中。由于 Web 信息数据量大，更新快，手工编写的方式缺点日益显现，不仅具有较低的抽取效率，而且需要花费大量时间和人力，很难满足大规模 Web 信息抽取的要求。随着人工智能技术的使用，在 Web 信息抽取中逐步引用了数据挖掘、机器学习和概念建模等技术方式，这些技术在一定程度上增加了程序的自动性，但也还是以用户的大量参与为基础的，自动化程度也不高。正是在这种背景之下，完全自动化的信息抽取方法逐渐成为 Web 信息抽取研究的热点，这些方法主要是分析网页的标签结构，挖掘网页中的重复模式来实现数据记录的全自动化抽取。而本体论的提出，又能解决抽取过程中需要用户干预的语义分析部分。

Web 数据集成系统是一个积累的系统。由于 WWW 在不断地发展，一方面，Web 数据会被不断地集成至系统中；另一方面，Web 数据间新的联系也将被集成至系统中。Web 数据集成系统中积累的数据，不仅可以为市场情报分析等应用提供支持，而且还可以为 Web 数据集成系统自身提供帮助，例如，利用系统中积累的数据辅助发现新 Web 页面中的待抽取数据，辅助发现新 Web 实体与 Web 实体模型中已有 Web 实体间的联系等。Web 数据抽取是 Web 数据集成中的关键问题开展 Web 数据抽取问题研究，对于 Web 数据集成具有重要的作用和意义。

（1）Web 数据抽取是实现 Web 数据集成的基础和保证

Web 数据集成需要对 Web 上的数据进行有效的整合。Web 页面中的数据大部分为半结构化数据，有效地对 Web 页面中的半结构化数据进行抽取，以规范化的形式存储，是 Web 数据集成的基本要求。Web 数据抽取可以完成对 Web 页面中广泛存在的半结构化数据的抽取工作，为 Web 数据集成奠定数据基础。

（2）Web 数据抽取可以实现对 Web 数据的理解

Web 页面中的数据大部分为半结构化数据，因此，Web 数据中存在大量的属性标签缺失现象，导致无法准确地理解对应 Web 数据元素的语义。Web 数据抽取可以对抽取到的 Web 数据元素进行语义标注，实现对 Web 数据的理解。

（3）Web 数据抽取为 Web 数据集成中的其他环节提供数据服务

Web 数据抽取可以利用已抽取的 Web 数据对象间的联系，发现 Web 实体间的潜在联系。在 Web 数据集成系统中，利用 Web 实体间的联系，可以形成一个基于这些联系的事实知识库，为进一步实施 Web 数据集成中的其他环节，例如重复记录探测、数据分析等，提供数据服务。

国内外学者对于 Web 数据抽取问题已经开展了大量的研究，取得了一系列的研究成果。但是，由于现有的大部分方法中侧重于从信息检索的角度展开研究，导致现有的 Web 数据抽取方法没有充分地利用 Web 数据集成系统中积累的数据。Web 数据集成系统中存在大量的数据，包含了大量的潜在特征和信息，有效地利用 Web 数据集成系统中积累的数据，可以进一步提高 Web 数据抽取的性能，为 Web 数据集成系统提供更多、更大的支持。

2. Web 数据抽取的问题

由于 Web 上的数据具有海量、异构、动态变化、联系丰富等特点，导致 Web 数据抽取研究中仍然存在以下问题有待解决。

（1）在 Web 数据集成过程中，需要获取 Web 实体的模式信息，为进一步识别、抽取和集成来自不同数据源的 Web 数据对象提供指导。Web 上广泛存在的半结构化数据的模式具有异构和动态变化的特点，有效地构建 Web 实体的模式信息，是一个有待解决的问题。现有的研究方法利用搜索引擎收集尽可能多网页上的属性标签以及 Web 数据对象信息，通过聚类的方法一次性地生成相关 Web 实体的重要属性标签集合。但是，由于 Web 上半结构化数据的模式具有异构性以及动态变化的特点，不同网站的网页上对同类 Web 数据对象的描述形式不尽相同，并且 Web 上会产生新的属性标签，导致该类方法无法满足 Web 数据集成的需要，无法根据 Web 数据模式的动态变化，实现对 Web 实体模式的逐步丰富。

（2）在 Web 数据抽取过程中，需要准确地从目标网页中抽取出目标数据，并对抽取的数据元素进行语义上的理解，为进一步整合数据奠定基础。准确地抽取目标数据，并进行语义标注是另一个有待解决的问题。

从目标网页准确地抽取 Web 数据对象是 Web 数据抽取研究中的主要研究点之一，通过对目标网站采样页面中的数据元素和属性标签进行准确识别，生成良好的训练样例，进而学习生成包装器，可以有效地实现对目标数据的抽取。现有的大部分方法主要利用采样页面自身包含的特征，例如视觉特征、结构特征等，识别出页面中的数据元素和属性标签，生成训练样例。但是，对于结构复杂的 Web 页面产生的训练样例质量不高，严重地影响了学习得到的包装器对于目标数据的抽取准确度。

Web 页面中包含的数据大部分为半结构化数据，存在大量的属性标签缺失现象，导致无法准确地理解对应数据元素的语义。条件随机场是目前进行 Web 数据语义标注的经典方法，但是现有

的条件随机场模型只能对 Web 数据元素间的邻接依赖联系进行处理，缺少对数据元素间的非邻接依赖联系的高效发现和处理机制，导致对于复杂页面的语义标注准确率较低。

（3）在 Web 数据抽取过程中，需要建立新发现 Web 实体与 Web 实体模型中已有 Web 实体间的联系，丰富 Web 实体模型，为进一步整合 Web 数据奠定基础。有效地建立新发现 Web 实体与 Web 实体模型中已有 Web 实体间的联系，也是一个有待解决的问题。

在 Web 数据集成系统中，需要将新发现的 Web 实体丰富至 Web 实体模型中，并建立新发现 Web 实体与 Web 实体模型中已有 Web 实体间的联系。利用 Web 实体间的联系，形成一个事实知识库，不仅可以为市场情报分析等应用提供支持，而且还可以为 Web 数据集成中的其他环节，例如重复记录探测、数据合并等提供强有力的支持。然而，现有的关系发现研究大多集中于命名实体间的关系发现，缺乏对 Web 实体模型中 Web 实体间联系的自动发现研究。

3. Web 解决方案

本书针对 Web 数据抽取中面临的问题，提出一组有效的解决方法，为实现 Web 数据集成系统提供必要的数据基础和数据保证，同时也为 Web 数据集成系统中的其他环节，例如数据整合层中的重复记录探测，数据合并等，提供数据服务。具体包括以下三个方面的内容。

（1）Web 实体模式构建

Web 实体的模式是指 Web 实体对应的 Web 数据对象所使用的属性标签的集合。通过利用 Web 数据集成系统中已存在的数据，辅助识别出目标页面中的新属性标签，并判断其所属类别，实现对 Web 实体模式信息的逐步丰富。

（2）数据抽取

Web 页面中的数据大部分为半结构化数据，利用 Web 数据集成系统中已存在的数据，辅助识别出目标页面中的数据元素和属性标签，提高目标数据的抽取准确率。另外，通过利用 Web 数据元素间的非邻接依赖联系，实现对 Web 页面中缺失属性标签的数据元素的准确语义标注。

（3）Web 实体间联系发现

Web 数据间具有丰富的潜在联系，Web 实体模型中定义了 Web 实体以及 Web 实体之间的联系。通过利用已抽取 Web 数据对象间的联系，建立新发现 Web 实体与 Web 实体模型中已有 Web 实体间的联系，实现对 Web 实体模型的丰富。

Web 实体的模式信息将为 Web 数据抽取提供指导，辅助识别页面中的待抽取数据元素以及确定页面中的待抽取数据区域。另外，Web 实体模式为 Web 数据语义标注提供必要的语义标签集合，辅助进行语义标注。Web 数据抽取提供的 Web 数据对象信息，可以为 Web 实体间联系的发现提供线索，辅助建立新发现 Web 实体与 Web 实体模型中已有 Web 实体间的联系。而 Web 实体的模式构建和联系发现又同属于 Web 实体模型丰富问题。

目前，针对 Web 信息抽取的方法很多，可以从不同的角度对这些方法进行分类，按照 Web 信息抽取对象的结构化程度，大体上可以分为三种类型：结构化文本、半结构化文本、自由文本。根据自动化程度可以分为人工方式的信息抽取、半自动方式的信息抽取和全自动方式的信息抽取

三大类；根据方法采用的原理可以分为基于自然语言处理方式的信息抽取、基于包装器归纳方法的信息抽取、基于本体方式的信息抽取、基于 HTML 结构方式的信息抽取、基于 HMM （Hidden Markov Model）的信息抽取、基于 XML 的信息抽取。本节将从方法采用的原理出发，介绍几种 Web 信息的抽取技术，并结合目前较为典型的系统来分析这几类信息抽取方法。

（1）基于自然语言处理方式的信息抽取

基于自然语言处理方式的信息抽取是以自然语言处理技术为基础，通常适用于那些含有大量文本且句子完整、适合语法分析的 Web 页面，在抽取的过程中，它将网页视为自由文本进行处理，经过句法分析、语义标注、专有对象的识别和抽取规则生成等一系列阶段。具体来说，它首先将网页文本分割成多个子句的集合，对每一个句子的句子成份进行语义标记。然后将其与已定制好的规则进行匹配，从而获得句子内容。对于规则的定制，我们可以利用知识工程的方法或自动学习的方法来获得。基于自然语言处理方式的信息抽取没有利用 Web 文档所固有的层次特性，在抽取过程中暴露出诸多的缺陷。首先其抽取规则的表达能力有限，缺乏健壮性，而且规则的获取是建立在对大量样本学习的基础之上的，不仅难以获得较高的抽取效率，而且难以实现完全自动化；其次，这种方式的信息抽取只支持记录型的语义模式结构，不支持复杂对象的抽取。

SRV 是一种用来生成抽取规则由上而下的关系算法，它将信息抽取视为分类问题的一种来进行考察。对输入的文档进行标记化，而且对所有连续的子串（如文本块）要么被标记为抽取的目标（正面实例），要么被标记为非抽取的目标（负面实例）。由 SRV 生成的抽取规则是一组依赖于面向标记特征（或预测分析）的逻辑规则。这些特征包含两种：简单类型和关系类型。简单类型的特征是一种功能函数，完成标记到诸如字符长度、字符类型等具体值的映射；关系类型的特征将一个标记映射到另一个标记，例如输入标记的上下文标记。学习算法开始于一组实例并动态添加预测以尽可能多地覆盖正面实例，尽可能少地覆盖负面实例。

（2）基于包装器归纳方式的信息抽取

包装器（Wrapper）实质是一个用于信息抽取的计算机程序，是针对某一特定数据源的抽取系统，由一系列的抽取规则以及应用这些规则的程序代码组成，其目的是用于数据的提取和分析，是信息集成系统中一个重要组件。包装器的任务是采用一系列规则，将用户所关心的信息从 Web 页面中抽取出来。包装器规则的生成依赖于原网页或其后台数据库的数据模式。一个包装器只能处理一种特定的信息源。它更注重于文本结构和表格格式的分析，针对性强，但可扩展性较差。由于一个包装器只能处理一种特定的信息源，所以若从几个不同的信息源中抽取信息，就需要一系列的包装器集。该类信息抽取不但对页面结构有所依赖，而且对页面内容也有所依赖，要想获得精确的抽取规则必须进行大量的样本训练。这样使得信息抽取的工作量巨大，同时也使其可重用性较差。应用包装器的信息抽取信息通常包括规则库、规则执行模块和信息转换模块。图 3.1 显示了应用包装器来进行信息抽取的模型架构。

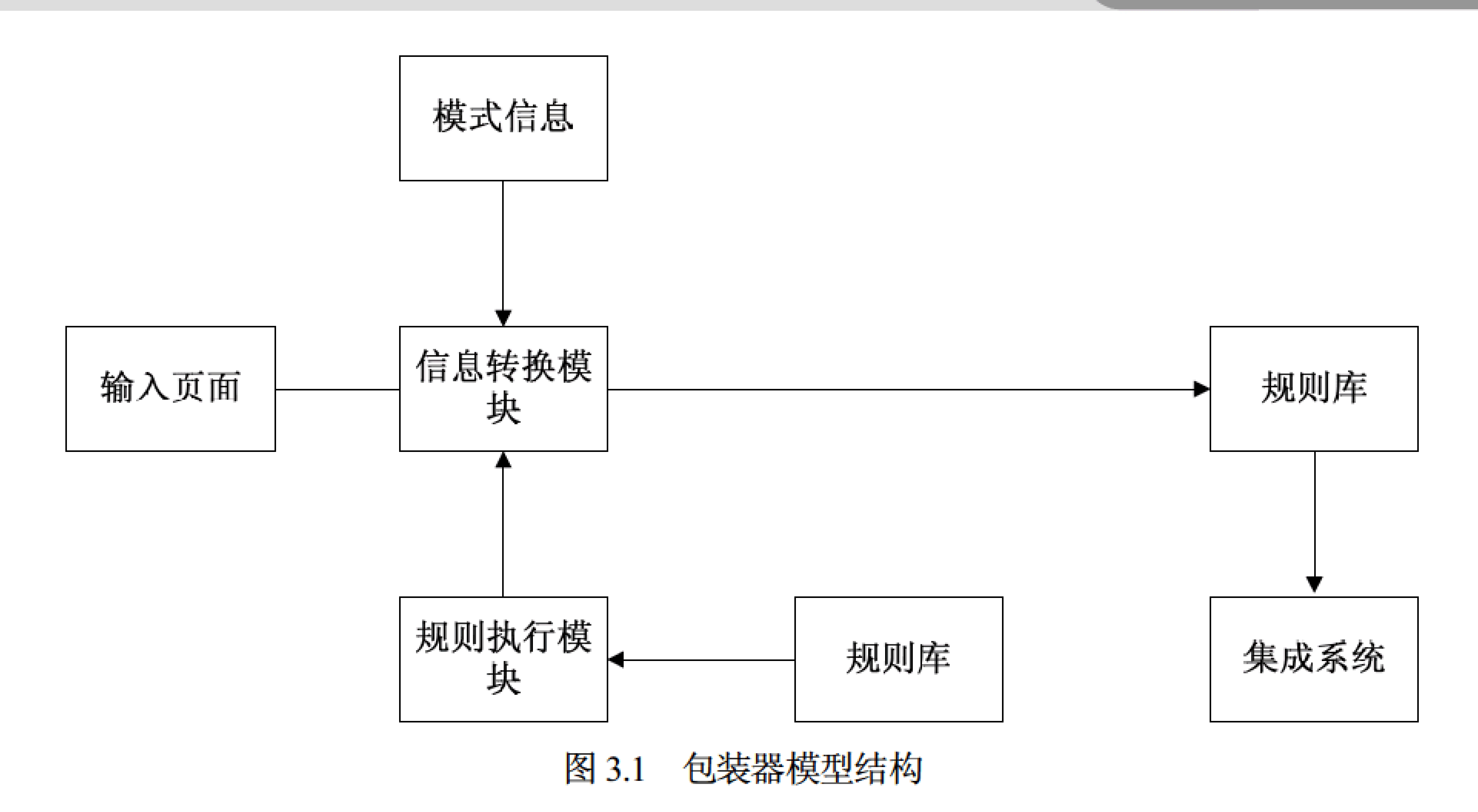


图 3.1　包装器模型结构

（3）基于 HTML 结构的信息抽取

Web 信息抽取的主要对象是 Web 页面，这些页面通常是由 HTML 标记语言进行编写的，具有非常清晰的层次结构。基于 HTML 结构的信息抽取便是利用了页面的结构来进行信息的定位。在此方式的信息抽取中，Web 页面通过转换器解析成反映 HTML 结构的 DOM 树，再通过某种方法把用户想要抽取的数据定位到该 DOM 树的层次位置上，然后利用正则表达式等匹配技术来得到些具体位置上的数据。该类信息抽取由于利用了 HTML 的结构特征，其抽取的准确率和召回率都比前两类抽取方法高，同时也适用于各个不同的知识领域。但同时由于现阶段网页的规范程度较差，HTML 页面的结构不够规范，这使得依赖网页结构的抽取方法对那些经常发生改变的目标网页变得不再可行。

（4）基于本体的信息抽取

基于本体的信息抽取是指将本体论应用到信息抽取中的一种抽取方法，它利用本体对数据本身的描述信息实现抽取。对页面的结构的依赖较少，因此容易处理网页结构变化的情况，这种方法的基础和核心是本体的构建，如何构造出良好的面向应用领域的本体对提高信息抽取的精确度有直接的影响。一般情况下，本体的构建是由领域知识专家通过对特定领域的调研分析采用人工方式定义而成的。本体中通常需要包括对象的模式信息、常值、关键字等。本体构建完成后，根据本体中的常值和关键字的描述信息产生抽取规则，对每个无结构的文本块进行抽取获得各语义项的值。

基于本体的信息抽取理论上说只要本体构建得足够的强大丰富，系统便能针对某个领域的网页进行准确的抽取而不用依赖页面的结构。但是本体的构建通常需要大量的专业知识和对领域的透彻理解，工作量极大。而且普通用户很难参与其中。

（5）基于 HMM（Hidden Markov Model）的信息抽取

HMM（Hidden Markov Model，隐马尔可夫模型）是最近应用最广泛的抽取知识表达模型，它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析。HMM 最初的应用之一是开始于 20 世纪 70 年代

中期的语音识别。HMM 成熟的学习算法和坚实的统计基础，使得它成为信息抽取中一种成功的模型。基于这种模型抽取技术的研究主要集中在两个方面。首先从数据中研究学习模型结构，这通常是以手工选定方式或从训练数据中学习得到；其次，根据给定的或者训练获得的模型对未标记的数据进行识别。其中将已标注数据用来训练 HMM 模型，未标记的数据结合到训练数据中来增大模型训练量，以学习更好的模型参数。

（6）基于 XML 的信息抽取

基于 XML 的信息抽取是运用 XML 的相关技术以 XML 格式的数据为数据源进行抽取处理。由于抽取的目标是 Web 网页，而 Web 页上的数据一般是以 HTML 格式存在的，所以它通常先将页面其转换为 XML 的格式再进行后续抽取。基于 XML 的 Web 数据抽取流程为：从 Web 网页上获得 HTML 格式的页面文档，然后将其转换为 XHTML 文件。再用编写好的特定 XSLT 文件对目标文件进行映射处理得到满足用户需求的 XHTML 文件；或为了使 XSLT 文件更加通用，需要先用 DOM 按照分析得来的算法对 XHTML 文件的节点进行过滤选择以缩小 XSLT 的映射范围，最后将该文档存储于结构化数据库中，如图 3.2 所示。

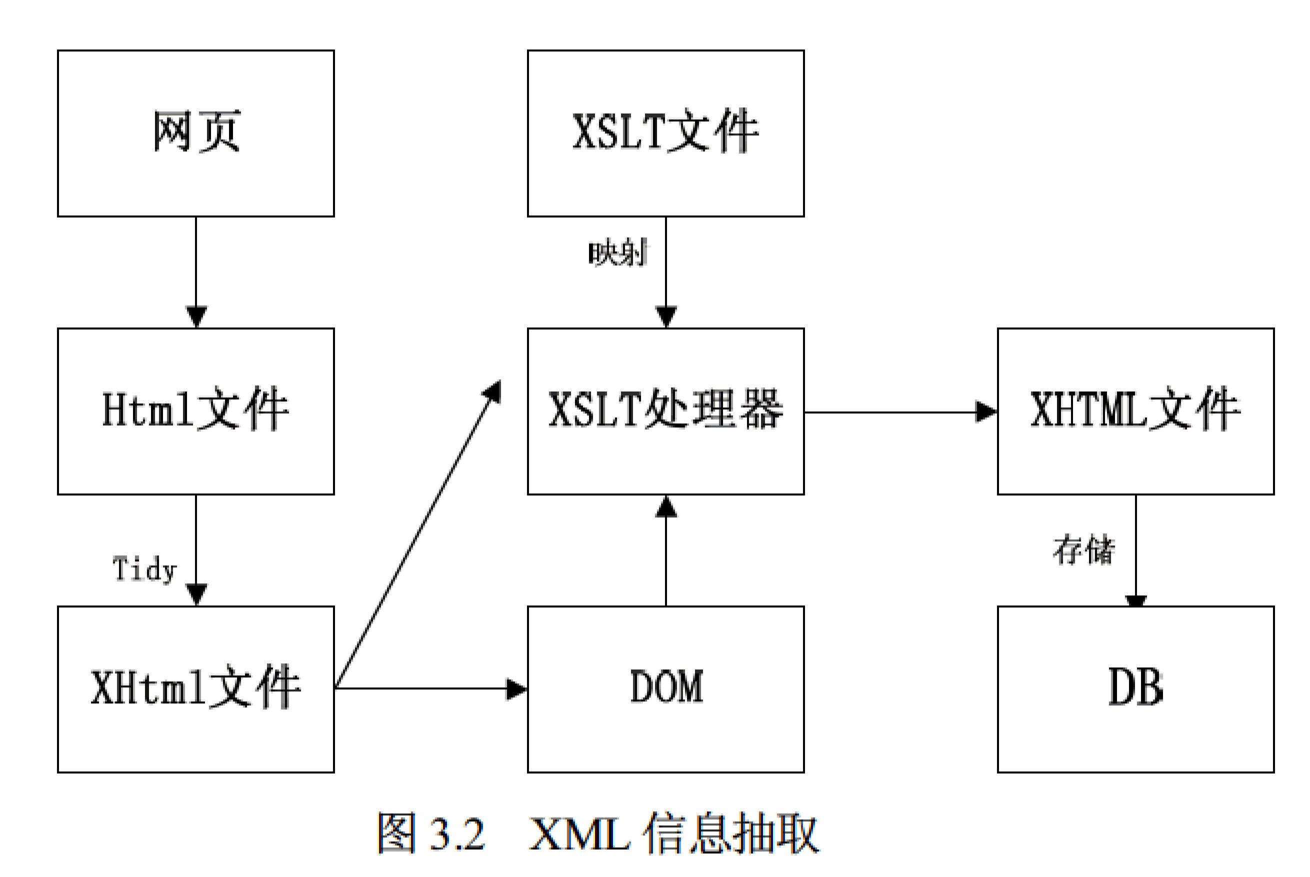


图 3.2　XML 信息抽取

4. 广域 Web 搜索

从 Web 上获取可靠的数据源是进行 Web 数据抽取的第一步。在寻求健壮解决方案的过程中，抽取可用的、最可靠和最稳定的数据源，使工作变得最简单。因而考虑以下这些要素非常重要：

- 数据源是否是在可靠的网络连接上生产可靠的数据?
- 数据源从现在起将存在多久?
- 数据源的布局有多稳定?

Web 上的数据变动性很大，有时结构也会发生改变，所以通常我们选择政府网站、大型商业网站、大型行业性网站作为 Web 数据来源。这也符合人们的习惯。人们为了获取更多明确的、实时更新的信息，往往是直接进入到一些特定的站点，进行站点内搜索。而且对于一个组织来说，因为其涉及如何有效搜索组织内部文档，Intranet 搜索的重要性也日益增加。

每天有成千上万的人通过像 Google 之类的搜索引擎在万维网上直接寻找信息，搜索引擎总会

把相关的结果返回给用户。广域 Web 搜索引擎的算法是通过分析 Web 页面间的链接结构而得来的。PageRank 算法和 HITS 算法是两种影响相当广泛的链接分析算法。许多学者在此基础上又提出了一些衍生算法。

（1）广域 Web 链接结构分析

Web 是一个超文本集合，且页面间的链接是有方向的，因此根据图论中有向图的定义，Web 可用称为链接图的有向图来建模。传统的方法是以页面粒度来对链接图建模：结点表示页面，有向边表示从一个页面到另一个页面存在一条或多条链接。由于一个 Web 文档通常是由其 Web 站点作者创作的多个页面链接而成的超文本文档，因此也可以用站点粒度来对链接图建模：结点表示站点，有向边表示从一个站点中至少有一个页面包含一个或多个超链接指向另一个站点中的某几个页面，此时称为站点图。

Web 虽然是一个分散的信息网络，但大量研究表明，Web 的链接结构具有自组织性：在全局上，信息源（页面或站点）间通过超链接按相同或相关的内容主题自然地聚合在一起，形成一个个群落，群落内部的信息源之间密集链接，而群落之间稀疏链接，甚至根本不链接；在一个群落内部，信息源之间的链接结构也有规律和特征，一个群落主要由两类信息源组成，一类主要被别的信息源所链接（包含高质量主题内容的信息源），另一类则主要链接到别的信息源（提供对高质量主题内容存取的信息源），它们均是用户想要的好信息源。Web 的这种自组织性为链接分析提供了依据，链接分析基于以下一个或两个简单的假设。

- 从页面 A 到页面 B 的一条超链接是页面 A 作者对页面 B 的一种推荐和称赞，意味着权威性或质量。
- 若页面 A 与页面 B 被一条超链链接，则它们可能有相同或相近的主题，意味着相关性。

大多数像 Google、A1taVista 这样的第二代搜索引擎已在其文档数据库中维护了页面间的链接信息，链接分析在 Web 信息检索中已普遍运用。

（2）广域 Web 搜索经典算法

经典的 PageRank 算法和 HITS 算法是两种应用相当广泛的广域 Web 搜索算法。

- PageRank 算法

PageRank 算法是最早并且最成功地将链接分析技术应用到商业搜索引擎中的算法。它的基本出发点是试图为搜索引擎所涵盖的所有网页赋予一个量化的价值度。每个网页被量化的价值通过一种递归的方式来定义，由所有链接向它的网页的价值程度所决定。显然，一个被很多高价值网页所指向的网页也应存在到其他网页的链接，那么这种网页将成为沉积网页，并使得上述这种转移的过程在沉积网页上永远终止。解决这个问题的方法很简单，假如一个随机冲浪者遇到了这种沉积网页，那么他可以随机地挑选另一个网页并继续他的浏览。为了对那些不是沉积的网页也一视同仁，这种类型的随机转移应该能以相同的概率在任何一个网页上发生。根据这种方法对网页排序以后，搜索引擎就可以决定以一种什么样的顺序将结果返回给查询用户。

- HITS 算法

HITS 算法是一种分析网络中网页间链接结构从而确定权威性网页和中心网页的链接分析算法，理解 HITS 算法是 Web 结构挖掘中最具有权威性和使用最广泛的算法。HITS（Hypertext-Induced Topic Search）算法是利用 Web 的链接结构进行挖掘典型算法，其核心思想是建立在页面链接关系的基础上，对链接结构的改进算法。HITS 算法通过两个评价权值——内容权威度（Authority）和链接权威度（Hub）——来对网页质量进行评估。其基本思想是利用页面之间的引用链来挖掘隐含在其中的有用信息（如权威性），具有计算简单且效率高的特点。HITS 算法认为对每一个网页应该将其内容权威度和链接权威度分开来考虑，在对网页内容权威度做出评价的基础上再对页面的链接权威度进行评价，然后给出该页面的综合评价。内容权威度与网页自身直接提供内容信息的质量相关，被越多网页所引用的网页，其内容权威度越高；链接权威度与网页提供的超链接页面的质量相关，引用越多高质量页面的网页，其链接权威度越高。

（3）广域 Web 搜索总结

可以对广域 Web 搜索的两种经典算法作进一步的探讨和比较。

PageRank 算法实质上是一种通过离线对整个互联网结构图进行幂迭代的方法。PageRank 所计算出的价值度的值实际上就是互联网结构图经过修改后的相邻矩阵的特征值。对这些值的计算有非常有效的方法（事实上，仅需要若干次的迭代计算即可以得到），因此能够很好地应用到整个互联网规模的实践中。这种方法的另一个主要优点是所有的处理过程都是离线进行的，因此不会为在线的查询过程付出额外的代价。但是，PageRank 算法存在一个大的问题，即价值度的计算不是针对查询的。对于某个特定主题的查询，在返回结果中一些与主题无关的网页将会排在较前的位置。这类问题对 PageRank 算法的影响则有必要作更进一步的研究。

HITS 算法在概念的定义上比 PageRank 算法多提出了一个中心性网页的概念。通过中心网页和权威网页的相互作用，HITS 算法更好地描述了互联网的一种重要组织特点：权威网页之间通常是通过中心网页而彼此发生关联的。HITS 算法和 PageRank 相似，也是通过迭代的方法计算相邻矩阵的特征向量。但 HITS 算法所针对的不是整个互联网结构图，而是特定查询主题的互联网子图。规模上的极大减小可以使 HITS 算法的迭代收敛速度比 PageRank 要快得多。但因为与查询相关，所以查询过程需要考虑排序的代价。另外，除非为 HITS 算法中所考虑的链接赋予适当的权值，否则，相邻矩阵的主特征向量并不能反映最合理的网页价值度排列。并且，即便对子图中的边赋予了适当的权重，如果子图的相邻矩阵是一个可约减的矩阵（例如图中有多个不连通的部分），那么很多有价值的网页仍将无法在主特征向量中得到体现。更为严重的是，在对很多广义主题进行查询时，HITS 算法会错误地将许多与主题无关的网页赋予很高的价值度，发生主题漂移。最后，应该注意到 HITS 算法所作用的查询子图是根据查询关键词在线构造的。通过常规的方法将无法满足在线查询响应时间的要求，但是，如果借助专用的连接服务器，查询子图的构建时间将是毫秒级的。

5. 小范围 Web 搜索

为了获取更多明确的、实时更新的信息，人们往往是直接进入到一些特定的站点对站点内进

行搜索。对于一个组织来说，Intranet 搜索的重要性日益增加，因为其涉及如何有效搜索组织内部文档。我们将发生在网络中一个闭合子空间的搜索定义为小范围 Web 搜索，常见的情况是在一个站点内搜索或者是 Intranet 搜索。现在应用于小范围 Web 搜索的产品生产方兴未艾，例如，AltaVista、Berkeley 的 Cha-Cha 搜索引擎等。然而人们多数使用的是广域 Web 搜索引擎，由于其并不能成功地运用于小范围 Web 搜索，所以对于小范围 Web 搜索的研究已提到了日程上来。我们分析小范围 Web 链接结构的特点，提出一种应用于小范围 Web 搜索的改进 HITS 算法，并通过矩阵理论以及实验证明是成立的。

现今存在的大多数小范围 Web 搜索引擎使用和万维网搜索引擎相同的技术，因而存在一些性能问题。根据一份 Forrester 调查报告显示，搜索引擎在某个站点进行搜索时，往往因为返回太多不相关的内容而不符合用户的需要。在调查中，对 50 个站点进行了测试，没有一个站点得到满意的结果。而且，小范围 Web 任务中，使用基于链接分析的方法几乎达不到预计的性能。这些都表明了现有的小范围 Web 搜索技术存在很大的弹性修改范围。

造成小范围 Web 搜索引擎低性能主要有以下几个方面的原因。

- 基于关键词相似度的查询方法存在一些缺陷。例如，查询词太短，使得匹配的空间很大，返回的页面虽然包含查询信息，但页面本身没有什么内容或者包含的内容价值不是很大。另外，查询词的一词多义，让搜索引擎将不同领域的内容混杂在一起提供给用户，显然这样的页面质量不高。
- 由于万维网的链接结构与小范围网络的链接结构并不完全一样，因而成功运用于万维网的链接分析技术不能直接运用于小范围网络。
- 虽然人们知道用户访问日志包含大量网页重要性的信息，然而，除了 DirectHit 之外，几乎没有其他的网站在这方面进行研究。

现在对于小范围 Web 搜索的研究已提到了日程上来，应用于小范围 Web 搜索的产品生产方兴未艾。大多数产品使用的是“全文本搜索”技术，这种技术针对查询检索出大量包含相同关键词的文档，然后根据关键词相似度对这些文档进行重要性排序。例如，AltaVista 提供了一个基于内容的站内搜索引擎；Berkeley 的 Cha-Cha 搜索引擎通过将搜索结果进行分类从而反映出未知的 Intranet 结构。M. Levenc 等人提出来的一种返回链接页面序列的浏览系统，帮助终端用户有效地浏览结果。这些系统都不使用链接分析技术对于搜索结果进行重要性排序，因而还存在相当大的空间来提高搜索的精度。

因为直接将链接分析技术应用于小范围 Web 搜索效果不佳，一些系统将它们的重点转移到用户的使用信息上。例如，DirectHit 就是一个典型的例子，它利用因特网上每天用户所作出的成千上万的决策来为搜索者提供相关度更高、组织更好的搜索结果。它是基于“点击次数”的概念，现在已被 Lycos、Hotbot、MSN、Infospac 等大约 20 多个搜索引擎使用。假设的前提是：根据某个特定的查询，若一个网页被访问的次数越多，则排序的权值越高。这种方法还有一些缺陷。首先，它需要大量的用户日志仅仅是为了少数的热门查询服务。其次，它很容易陷入到一个反馈死循环中，当访问一个热门网页时会使其权值增加，而权值的增加又使得该网页更加热门。

HITS 是基于链接结构，从而计算特征向量来分辨权威网页的链接分析算法。直观上来说，一

个有很多入度的网页拥有高推荐度，因而有较高的权威性。因而，链接分析算法有一个基本的前提：整个万维网被看作成一张推荐图，每一条链接代表一次推荐。也就是说，从网页 X 指向网页 Y 的一条链接意味着网页 Y 为网页 X 的作者所推崇。

对于万维网来说，链接分析算法的推荐假设在大多数情况下是正确的，因为链接确实在相当大程度上代表了作者的判断。当然也不排除某些不是为了推荐的目的而创建的链接，但是它们的影响基本上可以忽略不计。

然而，HITS 算法的推荐假设在小范围 Web 应用时却普遍不成立。在小范围 Web 中，大多数的链接比万维网中的链接要有规律。多数链接都是从父节点指向子节点，或者是由叶子节点指向根节点。其主要的原因是：小范围 Web 是由为数不多的作者所创建，链接创建的目的往往是为了组织站点内容使之成为一种继承或线性结构。因而，计算入度的方法并不能反映页面的权威性，使得已有的 HITS 算法不适用于小范围 Web 搜索。

在小范围 Web 中，我们将链接分为浏览链接和推荐链接两种。后者通常用于链接分析。然而，仅仅过滤出浏览链接远远达不到目的，因为剩余的推荐链接并不完全。换句话说，通过挖掘用户的访问日志可以在小范围 Web 中发现大量的间接推荐链接。

HITS 算法在小范围 Web 搜索的情况下表现并不令人满意，主要体现在：

- HITS 算法推荐假设理论不适用于小范围 Web 的链接结构，在小范围 Web 中，链接的构造只是为了组织整个 Web。
- HITS 算法整个的构建都基于初始集的选取，而初始集的扩充仅仅是经过了一次，因而出现遗漏的页面有很大几率。
- 算法所构造的输入数据为布尔型矩阵，其主特征向量并不能反映最合理的网页价值度排列。最后，如果图中包含了若干个不连通的子图，那么很多有价值的网页将无法在邻接矩阵的主特征向量中得到体现。
- 更为严重的是，在对很多广义主题进行查询时，HITS 算法会错误地将许多与主题无关的网页赋予很高的价值度。

归纳 HITS 算法应用于小范围 Web 的局限性主要有以下两个特征：图的不完整性和非加权输入。

针对上述分析，可以用一种适用于小范围 Web 搜索的改进 HITS 算法。该算法根据小范围 Web 的特殊链接结构，依据分解的用户浏览日志，首先构造含有间接链接的有向图，这样分解用户日志可以避免干扰数据，构造含有间接链接的有向图可以有效地解决图中包含了若干个不连通子图的问题，同时，可以清楚地发现页面间的关系，构造较完整的初始集。接着，将 HITS 算法的输入数据构造为两网页的链接权值矩阵，链接加权是根据用户访问网页的频率，从而达到不通过直接询问用户却合并用户反馈的目的。

6. Web 数据抽取过程

Internet 是一个巨大的信息仓库，包含丰富的信息资源和数据，这些数据多以 HTML 页面形式存在于不同的网站。然而 HTML 的设计目的是用于描述数据如何显示，而不是用来描述数据本

身，没有严格的结构限制和明确的数据类型定义。和存储在关系型数据库中的数据不同，这些存在于 HTML 页面中的数据是半结构化或非结构化的，无法直接利用。Web 数据抽取技术正是在这种背景下产生的。

Web 数据抽取的主要目的是从半结构化或非结构化的数据中抽取出特定的事实信息。比如从产品销售网站上提取产品的名称、规格、价格、数量等信息；从股票行情网站上提取股票代码、名称、价格、交易量等信息；从气象网站上提取城市代码、地名、天气、温度、湿度、风力、风向以及未来几天的预报等。当这些抽取出来的数据采用结构化形式表示之后，可以方便地保存到数据库、XML 文件供其他应用系统使用。

Web 数据抽取技术主要包括网页获取、抽取方法、抽取规则、数据校验与数据集成等内容。该技术的核心是从半结构化或非结构化数据中，识别出目标系统需要的数据，并将其转化为结构化、语义清晰的格式。

为抽取特定网站的 Web 数据，需要使用一定的方法，构建一个基于该数据源的适用的数据模型。虽然 Web 数据抽取模型的定义、结构和使用方式不同，但都是针对特定的一组页面进行处理，一般都称为网页包装器（Wrapper）。

包装器的构建是数据抽取技术的核心内容和关键技术。一般地，包装器由抽取规则（Extraction Rules）和抽取器（Extractor）两部分构成。抽取规则用于描述数据抽取方法，包括网页结构、数据项定位、抽取步骤、转换及校验规则，以及输出格式等。而抽取器是用于执行上述数据抽取规则的可执行程序或其他应用程序（如 XML 应用程序）。其中抽取规则是 Web 数据抽取技术的核心。

完整的 Web 数据抽取系统一般还包括网页下载及超链接分析、预处理、数据完整性校验、数据映射等，此外还包括数据集成（Data Integration）功能。

Web 数据抽取过程一般包括 5 个步骤：页面获取、数据抽取、数据校验及转换、数据存储、数据集成。

（1）页面获取

页面获取指通过指定一组 Web 页面的网址，或者根据指定的超链接导航规则，由抽取程序自动爬行并获取网页。有时还需要进行前置操作，比如当访问的页面有权限保护或其他限制时，往往需要执行脚本进行登录或其他认证之后才能访问页面。

根据数据抽取的特点，这些页面可分为两类：一类是数据页面，包含需要抽取的数据；一类是导航页面，包含指向数据页面的若干链接。典型例子是多页的产品列表页面。在很多情况下，数据页面中也包含了导航信息，此时两类页面在物理上组合在一起。有些网站的页面不仅仅使用静态链接，还使用复杂了 Web 表单或 JavaScript 代码的形式实现链接导航。对这种形式的导航页面，一般需要手工分析页面代码并编写相应的处理程序。

（2）数据抽取

获取页面之后，下一步就是数据抽取。数据抽取是使用包装器完成的核心功能，主要任务是根据抽取规则对于 Web 页面进行解析处理，抽取需要的数据项。在抽取之前，一般还需对 HTML 文件进行预处理，比如对 HTML 进行纠错和规范化，或者转换为 XHTML 格式从而以一致性更好

的 XML 方式进行处理。

（3）数据校验及转换

为了保证数据抽取的质量，减少无效数据及数据规范性，前一步骤中获得的数据一般还要经过数据校验及转换之后才能使用。数据校验的目的是确保抽取数据本身的正确性和完整性。对于明显的错误，可以通过正则表达式、校验规则文件等手段加以限制，并应用特定的领域知识和机器学习等技术进行纠错或将丢失的数据补充完整。数据转换的目的是保证数据的一致性。当从多个网站的数据源抽取并集成数据时，不同的网站可能会使用不同的命名规范、不同的表示方式以及不同的计量单位，即使在同一网站往往也存在这种不一致现象。此时应根据数据字典，将数据映射到统一的标准格式来保证抽取数据的质量，这可以应用领域知识，使用条件声明、规则表达式等技术来实现。

（4）数据存储

抽取得到的数据可根据应用系统的需求，采用不同的方式进行存储。如果得到的关系形式的数据可存储到关系数据库中；如果需要跨平台共享，可使用 XML 进行存储；如果需要以面向对象的形式存储数据，也可以用面向对象数据库的形式存储。还可以使用多种不同的数据持久化技术来存储。

（5）数据集成

抽取的 Web 数据来源于多个网站时，这些页面除格式不一致，数据组成也有不同。即使同一站点，也存在一组逻辑数据被分割到多个页面的情况，此时应进行数据集成。此外，页面中的数据一般是后台结构化数据的一个视图。例如，抽取的产品信息包含了供货商的名称、联系电话等信息，即数据存在冗余信息。这种情况需要应用领域知识，进行关系集合运算，对数据实体进行关系数据范式提升，以消除数据冗余及数据异常。

原型系统主要分为以下 3 个部分，自下向上分别为基础数据层、支撑服务层和功能服务层。

（1）基础数据层

主要用来存储原型系统中的各种类型数据。

① 爬虫爬取的网页以 HTML 格式进行存储。

② 元数据以 XML 格式进行存储，对 Web 实体模型和各种配置文件中包含的元素进行定义。

③ Web 实体模型以 XML 格式进行存储，Web 实体模型中定义了 Web 实体的模式信息以及 Web 实体间的联系。

④ 配置文件以 XML 格式存储，用以指导支撑服务层的基本服务能够自动地执行。

⑤ 网站注册文件以 XML 格式存储，记录了目标网站的信息。

⑥ Wrapper 和 Wrapper 注册文件以 XML 格式存储，其中，Wrapper 注册文件中包含各个目标网站对应的 Wrapper 注册信息，Wrapper 中存储了目标网站各组目标网页对应的 Wrapper，即一系列抽取规则。

⑦ Web 实体间联系文件以 XML 格式存储，记录了具有联系的 Web 实体对，以及相关的

Web 数据对象信息。

⑧ 采样数据以 XML 格式存储，记录了对各组目标网页进行采样抽取的结果，包括模式信息和 Web 数据对象信息。

⑨ 实际抽取数据以关系数据库格式存储，这些数据为课题组框架中的数据整合层提供数据基础，有待进行重复记录检测、数据融合等处理。

（2）支撑服务层

支撑服务层主要为功能服务层提供基础的原子服务，包括 Web 实体模型管理、元数据管理、配置文件管理、网站发现、Deep Web 爬虫管理、SurfaceWeb 爬虫管理、页面分类、页面标注、Wrapper 管理、Web 实体间联系发现及评价、Web 实体间联系产生和数据服务。

① Web 实体模型管理包括新建模型和丰富模型两个部分。新建模型主要用于为新发现的 Web 实体建立 Web 实体模式；丰富模型主要用于对 Web 实体模型中已有的 Web 实体模式进行丰富。

② 元数据管理主要用于定义 Web 实体模型和各个配置文件中的基本数据项。

③ 配置文件管理包括建立配置文件、读取配置文件、修改配置文件和删除配置文件。其中建立配置文件是在元数据的指导下完成的。

④ 网站发现主要用于获得待抽取的目标网站，网站发现方法分为两类，一类由业务专家手工设定；另一类是设计自动发现方法，自动对新网站进行评估和选择。

⑤ Deep Web 爬虫管理是指，针对 Deep Web 网站，采用有效算法自动迭代地构建有意义的查询词，通过查询接口提交查询，爬取 Deep Web 页面，爬取后的 Web 页面以 HTML 形式进行存储。

⑥ Surface Web 爬虫管理是指，针对 Surface Web 网站，利用普通爬虫爬取待抽取的目标页面，爬取后的页面以 HTML 形式进行存储。

⑦ 页面分类主要对从目标网站爬取的网页根据待抽取 Web 实体的类型以及网页模板类型进行分类，以便学习生成包装器，进行数据抽取。

⑧ 页面标注主要用于对采样页面进行标注，识别出 Web 数据对象中的数据标签和数据元素，为构造包装器提供训练样例。

⑨ Wrapper 管理包括生成 Wrapper、注册 Wrapper、选择 Wrapper 和执行 Wrapper。其中生成 Wrapper 是指利用目标网站中的训练样例，结合现有的 Wrapper 生成技术，生成目标网页对应的 Wrapper。注册 Wrapper 是指将各个 Wrapper 的信息写到 Wrapper 注册文件中，目的是当执行大规模数据抽取时，系统自动地根据 Wrapper 注册文件中的注册信息，为各个网站目标网页选择对应的 Wrapper，执行大规模数据抽取：选择 Wrapper 是指在 Wrapper 注册文件中检索出目标网站中目标网页对应的 Wrapper；执行 Wrapper 是指根据 Wrapper 中的抽取规则，自动地抽取出目标网页中的待抽取数据。

⑩ Web 实体间候选联系发现与评价主要用于发现 Web 实体间的候选联系，并利用相关评价指标对候选联系进行评价，将具有高置信度的候选联系返回给业务专家。

⑪ Web 实体间联系产生是指在业务专家的参与下，完成对 Web 实体间联系的确认和描述，同时将新发现的 Web 实体间联系丰富至 Web 实体模型中。

⑫ 数据服务主要用于对不同类型的数据（HTML、XML 和关系数据库等）进行访问和操作，支撑服务层的其他所有服务均需要通过数据服务来访问基础数据层的数据。

（3）功能服务层

功能服务层是该原型系统从逻辑层面提供的功能服务，包括网页爬取、采样抽取、大规模抽取和 Web 实体间联系发现。

① 网页爬取首先调用支撑服务层的网站发现，获得目标网站名称。然后，根据目标网站的类型，分别选择 Deep Web 爬虫管理服务和 Surface Web 爬虫管理服务完成对目标网页的获取。

② 采样抽取通过读取相应配置文件中的信息，例如，采样网站名称、采样 Web 实体名称、采样网页个数等，获取采样规则；然后，利用页面标注服务对目标网站中的采样页面进行自动标注；最后，对采样网页进行抽取，得到采样网页中关于待抽取 Web 实体的模式信息和具体 Web 数据对象。同时，利用 Wrapper 管理中的 Wrapper 生成服务，学习得到目标网站对应页面的 Wrapper，写入至 Wrapper 注册文件。

③ 大规模抽取通过读取相应配置文件中的信息，包括目标网站的名称、待抽取的 Web 实体名称；然后，利用 Wrapper 管理中的 Wrapper 选择服务，为目标网站目标网页选择对应的 Wrapper；最后，利用 Wrapper 完成目标网页的大规模数据抽取。抽取的数据存储至实际抽取数据库中。

④ Web 实体间联系发现通过调用 Web 实体间候选联系发现与评价服务，获得高置信度的候选联系；然后调用 Web 实体间联系产生服务，由业务专家完成联系的确认和说明；最后将产生的新联系丰富至 Web 实体模型中。

7. 小结

近年来，Web 数据集成研究已经成为一个很重要的研究课题。Web 数据抽取是 Web 数据集成中的关键问题，由于 Web 数据具有大规模性、异构性、动态变化和关系丰富等特点，使得从 Web 页面中准确地抽取出广泛存在的半结构化数据成为当前的热点研究问题之一。可以以 Web 数据集成为目标，针对 Web 页面中的半结构化数据抽取，通过充分利用 Web 数据集成系统中已存在的数据，有效地解决了 Web 实体模式构建、Web 数据准确抽取和语义理解，以及 Web 实体间联系发现问题。

Web 数据抽取是一个充满机遇与挑战的研究领域，还有许多问题有待于进一步的探索与研究。需要注意的工作如下。

（1）Web 数据抽取的适应性问题

由于 Web 上数据具有动态变化的特点，网站和 Web 页面上的内容经常发生变化，导致已产生的抽取规则失效。如何有效地提高 Web 数据抽取的自适应能力，使之能够根据目标网页发生的变化自动做出调整，更新相应的抽取规则，准确地完成抽取工作对于 Web 数据集成变得至关重要。下一步将针对保持 Web 数据抽取的适应性问题，提出有效的解决办法，增强 Web 数据抽取系统的适应能力，满足 Web 数据集成的需要。

（2）Web 数据抽取的通用性问题

目前，大部分 Web 数据抽取方法是针对单一网站构造相应的抽取系统，完成准确的数据抽取工作。但是，由于 Web 数据集成系统需要对互联网上的众多网站上的内容进行集成，因此，导致构建 Web 数据抽取系统的代价也非常高。如何有效地构造通用性比较强的 Web 数据抽取系统，使之能够同时完成对多个网站的数据抽取工作，可以极大地提高 Web 数据抽取的效率。下一步，将针对提高 Web 数据抽取的通用性问题进行探讨。

（3）Web 页面中的非结构化数据抽取问题

针对非结构化文本的信息抽取技术，是自然语言理解领域的一个重要研究课题。中文信息抽取方面的研究起步较晚，主要的研究工作集中在对中文命名实体的识别方面，在设计实现完整的中文信息抽取系统方面还在探索阶段。Web 页面中也存在着非结构化文本，如何对 Web 页面中的非结构化数据进行抽取，并进行相应的语义理解，对于 Web 数据集成也有着至关重要的意义。

3.3.2　非结构化数据抽取

1. 非结构化数据采集的背景和意义

计算机应用的不断发展导致了数据量的急剧增加，由于数据结构化过程受限于人工处理速度，导致非结构化数据的增长速度远远大于结构化数据。传统上使用文件目录树组织管理大规模非结构化数据的方案存在很大缺陷。文件目录树不能很好地表达非结构化数据自身以及数据之间语义关系的多样性，同时在大规模数据集下维护文件目录树的一致性会非常困难而且开销极大。因此，对海量非结构化数据的组织进行研究，成为如今迫在眉睫的问题。

随着计算机技术不断深入到社会生活的方方面面，数字化信息以指数规模增加。在如此巨量的数据面前，人对数据的处理速度是非常慢的，远远落后于计算机对数据的处理速度。但对于数据的深层理解和应用，现有的计算机系统却暂时无法有效地实现。因此使用高效的计算机数据处理手段加快人类数据处理和应用成为亟待解决的问题。

在现实生活中，数据的获取速度往往快于人的处理速度，因此这些获取的数据必须进行存储，以待未来使用或者应用，而高效的数据组织方法能够帮助人们在需要时迅速地从后台大规模数据中获取自己想要的数据；另一个方面，对于数据的理解往往表现为一个不断深入的过程，在此期间必需保存数据自身和对于数据的不断认知，如何存储这些新增的数据成为当前需要解决的问题。

现代计算机对数据的管理通常以文件的形式，通过文件系统来对数据进行组织管理。传统的文件系统包括两个层次的操作语义，第一相对于用户而言，以文件目录树的形式提供文件（数据集）逻辑组织的方法，目录和文件结构包含一定的逻辑含义；另一个方面，文件系统和存储设备之间使用块操作语义，其中最为关键的结构实现文件的分配信息表和文件目录结构的存储表示方法。其中文件目录结构既起到数据信息关联表示的作用，又起到寻址作用（通过上级目录定位下级目录或者文件的分配信息结构）。

传统的文件系统不能够很好地解决大规模数据的组织问题。主要表现在以下两个方面。

从用户的角度来看，文件目录树不能很好地表达数据之间的逻辑关系，由于一级目录和子目录、子文件之间的关系一般是包含或值分类关系，当涉及文件的多个属性时，就很难根据一个属性进行分类。例如文档有生成时间、作者等属性，那么是按照时间建立目录结构，还是使用作者进行分类就不能确定。因此，文件目录树不能很好地表示文件之间的关系；从存储过程来看，树形目录包含从根到子树之间的遍历关系，当根或者上级目录失效时，下级文件或者目录就是不可以访问的。而且上级目录也可能成为性能的瓶颈。在本地系统中，也许不是很大，当文件目录树扩展到多个结点的分布式系统中，维护树的一致性就会很困难，而且从根遍历也会导致访问多个结点，造成不必要的开销。

从数据检索的角度来看，传统的文件系统由于记录信息的元数据缺少灵活性，无法充分体现用户对信息的理解，而用户对信息的理解是用户使用特性和信息实体化分类的综合体现。基于人类行为学的理论，通过用户使用特性可以很好地理解用户处理信息的一般性动作，实现信息实体化分类。因此，在海量非结构化数据检索中由于信息存储本身缺少足够的表征用户偏好信息和信息实体化分类标记的元数据，从而导致信息检索的精度较低、代价较大。因此，寻找更为合理有效的海量非结构化数据组织和管理的方法机制，就成了迫切的任务。

2. 非结构化数据和结构化数据的比较

（1）结构化数据组织

数据库是现代计算机信息系统和计算机应用系统的基础和核心技术，同时也是对结构化数据组织管理最为适合的方式。数据库技术最初产生于 20 世纪 60 年代中期，根据数据模型的发展，可以划分为三个阶段：第一代是网状和层次数据库系统；第二代是关系数据库系统；第三代是以面向对象模型为主要特征的数据库系统。

第一代数据库的代表是 1969 年 IBM 公司研制的层次模型的数据库管理系统 IMS 和 20 世纪 70 年代美国数据库系统语言协会 CODASYL 下属数据库任务组 DBTG 提议的网状模型。层次数据库的数据模型是有根的定向有序树，网状模型对应的是有向图。这两种数据库奠定了现代数据库发展的基础。这两种数据库具有如下共同点：

- 支持三级模式（外模式、模式、内模式）。保证数据库系统具有与程序的物理独立性和一定的逻辑独立性；
- 用存取路径来表示数据之间的联系；
- 有独立的数据定义语言；
- 导航式的数据操纵语言。

第二代数据库的主要特征是支持关系数据模型（数据结构、关系操作、数据完整性）。关系模型具有以下特点：

- 关系模型的概念单一，实体和实体之间的连接用关系来表示；
- 以关系代数为基础；
- 数据的物理存储和存取路径对用户是透明的；
- 关系数据库语言是非过程化的。

第三代数据库产生于 20 世纪 80 年代，随着科学技术的不断进步，各个行业领域对数据库技术提出了更多的需求，关系型数据库已经不能完全满足这些需求，于是产生了第三代数据库。主要有以下特征：

- 支持数据管理、对象管理和知识管理；
- 保持和继承了第二代数据库系统的技术；
- 对其他系统开放，支持数据库语言标准，支持标准网络协议，有良好的可移植性、可连接性、可扩展性和互操作性等。第三代数据库支持多种数据模型（比如关系模型和面向对象的模型），并和诸多新技术相结合（比如分布式处理技术、并行计算技术、人工智能技术、多媒体技术、模糊技术），广泛应用于多个领域（商业管理、GIS、计划统计等），由此也衍生出多种新的数据库技术。

分布式数据库允许用户开发的应用程序把多个物理分开的、通过网络互联的数据库当作一个完整的数据库看待，具体可以通过客户/服务器模型来实现分布式数据库。并行数据库通过 cluster 技术把一个大的事务分散到 cluster 中的多个节点去执行，提高了数据库的吞吐率和容错性。多媒体数据库提供了一系列用来存储图像、音频和视频对象类型，更好地对多媒体数据进行存储、管理、查询。模糊数据库是存储、组织、管理和操纵模糊数据的数据库，可以用于模糊知识处理。

（2）非结构化数据组织

随着科学技术的发展，网络的普及，数据库的弊端便逐步显示出来，最严重的莫过于对半结构化、非结构化数据处理的捉襟见肘上。在对非结构化的数据组织管理上，存在文件目录树、索引及检索、语义文件系统几种方法。

① 文件目录树

在计算机内部，数据和信息通常以文件的方式来进行存储及组织管理，对数据的组织与管理，也可以看成为对文件的组织与管理。目前，目录树是最常用于文件管理的结构。目录树通过文件的路径名来对文件进行分类管理，其优势是用户通过其对文件内容的理解，来建立路径名，将文件精确地存放到某个路径中。文件的路径名包括了逻辑语义和物理地址的作用，即用户通过文件路径名来进行逻辑管理，而操作系统通过文件路径名来对文件进行块操作语义。随着文件数目的不断增长，这种管理方式就变得低效。

在使用传统的文件目录树方式来管理海量非结构化数据时，用户会很容易就陷入两难境地：既想将文件更加详细的分类，又无法记住标记文件详细分类的绝对路径名。因此，海量非结构化数据不适宜使用传统的文件目录树方式来管理和组织数据。

② 索引及检索

索引最早出现在在文献系统中，从这个意义上讲，索引是指文献集合中包含的事项或从文献集合中引出的概念的一种系统指南。这些事项或引出的概念是由按已知的或已说明了的可检顺序排列的款目表达出来的。由于计算机的出现，索引技术在现代得到了迅速的发展，特别是数据库系统中的索引技术。“索引”在数据库的术语中是指根据某特定域（或属性）对数据库中数据的一种排序，这一特定域（或属性）称为关键域或关键属性。对应的索引服务就是根据索引从数据中

提取信息，再对这些信息进行有效的组织、分析后，提供给用户。数据库中采用了类似于看书查目录一样的索引技术，使得查询时不必扫描整个数据库就能迅速查到所需要的内容。

在现实情况下，绝大多数的用户不愿意也无法记住所有数据信息的绝对路径，或者说所有属性。但是用户能提供的是用于描述所需要的数据信息的某些特征，用户也希望能够利用这些少量的特征信息，来缩小数据文件集，从而更迅速地定位到自己需要的数据。目前大多数的文件系统，可通过建立一些特定的索引，给用户提供相关的检索工具。

当前的索引及检索工具的好处就是用户仅仅需要一些简单的操作，就可以快速地找到部分需要的数据。但是其也有很大的缺陷，因为提供的索引都是预先定义或者是对文件绝对路径分析而得到的，准确性和易用性都无法满足在海量非结构化数据下的使用。

③ 语义文件系统

为了摆脱层次化文件目录树的缺陷，很多机构研发出了基于属性的索引方法。BeFS 提供了另外的数据结构来通过属性来对文件集进行索引，通过<关键词，文件>对并且建立对应的索引，用户通过这种索引结构，提供关键词来快速寻找相关的文件。

语义文件系统通常使用<分类，值>来给文件赋予可检索的映射，分类也可以称为文件的属性，属性可以通过用户输入或者其他方法来获取，例如对全文分析、对文件路径的数据提取，等等。一旦属性建立，用户就可以建立该属性的虚拟文件夹，所有包含该属性的文件都可以链接到这个虚拟文件夹下，如果属性与属性之间有继承关系，那么父属性可以通过虚拟子文件夹的形式来体现。

海量数据的“海量”，主要表现在两个方面：第一是在存储容量上，所谓的海量存储一般指存储容量超过 PB 级的大规模存储；第二就是在数据结构上，前者的海量就决定了数据源的多样化，即这些数据会以网页、语音、图片等离散形式存在，这些数据本身是半结构化、无结构化数据。因此，海量非结构化数据组织方式就不能简单地采取传统的文件方式、数据仓库方式、主题树方式和超媒体方式，而对海量非结构化数据的组织，也有其特定的需求。

- 用户参与：由于计算机对数据的理解能力非常弱，人的理解能力很强，而且海量非结构化数据的获取速度远远大于对其的处理速度，因此对数据的理解是一个渐进的过程。只有通过人的参与才能够更加精确地抽象出数据的特征，从而用这些特征来描述数据。
- 自动化：由于海量非结构化数据的种类繁多、数量庞大，以手工方式进行处理已不能满足海量非结构化数据组织的需要，必需采用自动化的数据组织手段。
- 模式提取：数据之间存在着多维的联系，这些联系可以作为数据的特征值来表征处于联系的端点上的各种数据。特别是在海量非结构化数据环境下，抽取这些联系能够降低数据规模。这些联系可以认为是知识，一种经过验证、完善后的形成的知识。这就是模式。半结构化或者无结构化的数据没有事先固定的模式，但可以从元数据中归纳出反映当前状态的结构模式。这个过程就是模式提取。
- 语义集成：海量非结构化数据组织中的数据领域种类也是很多的，针对每个领域生成领域内各信息的语义网络并建立有效的集成框架。这样可以解决各个异构信息源之间的语义不一致性，以提供基于语义的操作与服务。

上面的需求决定了海量非结构化数据组织必须要在文件系统、模式识别、数据挖掘等方面全面考虑。

3. 海量非结构化数据组织设计

由于计算机中的数据具有积累性、沉淀性的特点，人对事物的认识是渐进性的，是一个从简单到丰富、从具体到抽象的过程，结合语义文件系统和索引机制的特点，对海量非结构化数据的组织管理，应该具有：

- 逻辑分类与物理存储分离；
- 通过<属性，值>来描述数据的特征；
- 对描述数据的<属性，值>集不应该有限制，因为人对事物认识的渐进性，随着用户对数据信息认识的不断加深，对数据信息的描述也会不断丰富；
- 根据不同的<属性，值>，重新进行数据组织，产生新知识；
- 更加高效的索引及检索机制；
- 根据用户的行为习惯，更方便高效地将用户所需要的数据呈现出来。

海量非结构化数据组织与管理在设计过程中，都围绕着如何更加合理智能地产生<属性，属性值>从而体现出用户对信息认识的渐进性，如何让使用者更加快速准确地获取自己需要的数据等而展开。

根据海量非结构化数据组织的需求，功能模块划分为 5 大部分：一为属性获取模块，负责<属性，属性值>值对的生成、修改、删除以及一致性的相关操作；二为属性组织模块，负责对存在系统中的属性进行组织关联，形成属性关系网；三为 THLI 模块，负责生成索引以及提供检索功能；四为逻辑视图模块，负责对结果数据集进行在分类和产生热点导航；五为 XML 数据库操作，负责对 XML 数据库的相关操作。

当数据以文件的形式进入文件系统时，可以对文件进行属性处理，通过系统对文件的属性以及系统原有的属性集进行再组织后，使用 THLI 机制对属性进行索引；用户进行检索时，输入属性和属性值，先通过 THLI 检索是否存在相关的属性，然后再在返回的结果集内检索符合属性值的数据集，最终呈现给用户。数据模型（属性，属性值，关系）通过 XML、数据库进行存储。

4. 非结构化数据的数据模型

（1）数据对象

数据对象与文件系统的文件对应，数据对象包含有描述该数据的属性和属性值，通过<属性，属性值>值对来进行组织。

（2）属性

属性用来描述文件的特征，同时通过属性来将文件进行分类。属性包括系统属性和扩展属性。

- 系统属性：大多数文件系统提供了一部分属性来描述文件的信息，例如文件的创建时间，最后修改时间、文件名、路径、权限等，这些属性又被称为元数据。文件系统通过这些元

数据对文件进行组织，用户可以根据某个或者多个元数据的信息来对文件系统里的文件进行查询检索。

- 扩展属性：用于更详细描述文件特征的信息，由于系统提供的属性大多是为了方便操作系统进行管理，而且是通用的，无法详细描述各个文件的特征信息，所以考虑了属性的扩展，由于不同文件之间的差异性可以很大，随着用户对文件载有信息理解的不断加深，对文件的特征描述也会越来越丰富，因此扩展属性不能像文件系统一样，把元数据固定个数和格式，存放在底层的数据结构中。

（3）属性值

考虑到词汇意思的多样性，以及方便表示具有相似属性而实际又存在着个体差异的文件聚类，可以增加属性值，通过<属性，属性值>作为一个扩展属性元组来描述文件。例如，一个描述红绿灯的文件可能会有颜色的属性，属性值是红色、黄色、绿色的一个属性值集合。考虑加入属性值，通过<属性，属性值>的扩展属性元组来描述信息，是因为其比单单采用属性作为关键字描述信息更加灵活和方便，同时更能清晰地反映出信息的特征。

（4）关系

属性与属性之间通过关系联系在一起，引入关系后，在实际意义上存在关系而在语义上毫无关联的属性就可以联系起来，属性与属性之间的关系具有权值，通过权值来判断两个属性的联系紧密度。这样，属性、关系构成了一张带权值的网状图。

5. 属性的获取策略

属性的获取策略，即通过某些方式来生成用于描述文件特征的属性。从语义文件系统中可以借鉴和采纳很多非常有用的方法。属性的获取策略有两种，一种是直接获取，另外一种是间接获得。直接获取是在文件生成或者被用户访问的时候产生；而属性的间接获得是通过系统对文件空间、时间等关系或者用户的访问模式的分析推断而产生。

在文件生成或者被访问时，可以通过几种方式来生成描述该文件的属性。

- 用户输入：由于用户对数据的理解能力，是计算机无法模拟的，因此提供用户直接输入描述数据的属性的方法是必要的。随着用户对数据信息的不断理解，可以通过这个接口来丰富描述信息，即<属性，值>。用户可以对信息做出自己的理解，从而给出信息的新的自定义属性和属性值。
- 上下文分析：上下文是“存在或发生的事物的一些内在关系”，例如，一个文件的上下文包括与该文件同时进行存取的文件、当前用户进行的工作，甚至用户所在的物理位置，用户对该文件进行的所有动作和关联的数据都可以看成是该文件的上下文。大多数用户通过上下文来对文件进行组织以及检索。例如，用户可能将当前特定任务的所有相关文档归类到一个单独的文件夹中，或者通过回忆同时期访问过的其他文件来检索自己需要的文件。这些语义上的关系，不可能从文件本身的内容获取。通过对文件的上下文进行分析，也可以为文件赋予某些相关的属性和属性值。

（1）基于关系的策略

- 空间关系：在原有的文件系统中，用户通过文件目录树的层次关系来组织管理文件，同一个文件目录下的文件，其在某种分类上是具有相同或者相似的特征的。而与其父目录下的文件，存在的是特征继承的关系。因此，通过对空间关系的分析，在原有的文件目录树层次模型的方式下，对同层次的文件集考虑赋予相同的属性，对父目录的属性进行继承。
- 时间关系：承载数据信息的文件，其内容并不是一成不变的，用户可能在某个时间段对其内容进行修改，从而描述这个文件特征的属性也不应该一成不变，而是应该能够反映出时间关系。在时间轴上有关系的一些文件，其特征也会有一定的关系。
- 用户的访问模式：在用户访问文件集时，访问的模式隐含了被访问文件之间的联系，很多技术已经利用这种方法来提高文件访问的性能 （例如文件的预取、缓存技术）。

（2）属性的存储策略

属性的数据与传统的数据库数据不同，传统的数据库都有一定的数据模型，可以根据模型来具体描述特定的数据。而属性的数据是半结构化的，几乎没有特定的模型描述，每一个文件由于其内容的差异性，用于描述其特征的属性数据就不会一致，除系统属性是所有文件都拥有的信息，具有一定的结构性之外，其他的扩展属性就体现了异构性，因此属性的数据是非完全结构化的，这也被称为半结构化数据，与 Web 上的数据非常相似。半结构化是属性结构的最大特点。

传统的关系数据库本质上采用的是一个二维的模型，通过一系列二维关系的组合来描述复杂实体对象，每个表所代表的所有实体在建模设计时没有差异性，即使只有一个实体不拥有某种属性，但为了使其结构化，也必须为其建立一个字段。如果这种个体间的差异性不是仅仅表现在属性上，而且涉及结构和关系，则需要为有差异的实体建立不同的表和对应关系。处理无结构化和半结构化的数据，是传统的关系型数据库的天生缺陷。因此，传统的关系型数据库不适合用来表示属性，已经对属性信息进行存储。由于 XML 在半结构化数据表示上的优势，因此也可以用来描述属性信息。

在原型系统中，采用了 XML，来对数据模型描述，使用 XML、数据库及其提供的 API 来进行 XML 文件的存储管理。所有数据模型与 XML 对应的格式如下。

- 对象：每个对象对应于一个文件，包含系统属性和扩展属性。系统属性是格式化数据，每个对象都拥有相同的系统属性，但其属性值不同。对象的 XML 文件名通过文件的绝对路径去掉特殊字符后取对应的 char 值表示。
- 属性：每种属性都有一个对应的 XML 文件，XML 的文件名为对应的属性名字。记录了该属性的一些统计信息，例如引用次数、被检索次数，记录了该属性有的属性值，以及引用了该属性的文件的链接。在原型系统中，属性对应的 XML 文件可用于反向索引，通过该反向索引可以快速定位到拥有该属性的文件信息。
- 关系：每种关系也会有对应的一个 XML 索引，与属性的索引一样，记录了用户的一些行为特征，包括检索次数和引用次数。同时记录了有该关系的属性对，这种<属性，属性>对是有方向的，而且是带权值的。

3.3.3 基于云计算的海量数据分析

随着信息技术的高速发展，如今每 18 个月产生的数据量大约等于过去几千年产生的总和，并且有不断增加的趋势。如此多的数据无疑能为人们带来广阔的信息量，但需要从海量数据中发现对企业或个人有用知识的难度随之增加。而云计算平台能够进行动态资源调度和分配，具有高度虚拟化和高可用性等特点，正好能满足高效数据抽取和挖掘的需求。将云计算技术与现有的数据挖掘技术进行有效结合不失为一种可行的途径。云计算是一种能够通过互联网为用户提供服务的计算模式，它提供的主要是能够进行动态伸缩的虚拟化的资源，用户不需要了解如何管理那些支持云计算的基础设施。简单来说，云计算是一种新颖的商业模式，它使用大量廉价的、相互连接在互联网上的计算机进行任务的处理，为各种应用系统提供所需要的存储资源、计算资源和其他服务资源。

从技术层面来说，云计算技术早已存在，它是虚拟化技术的扩展、分布式计算技术的演进、SOA 架构的延伸、信息资源的集中管理和智能调配机制的体现。与传统 IT 技术有所区别的是，云计算带来了理念创新。从商业角度来看，云计算的核心理念是以服务的形式提供计算资源，用户在需要时进行使用和购买，可以更好地满足组织业务快速变更和创新升级的需求。对于像数据挖掘任务这种大量的数据密集型应用，往往需要牵扯到近似求解、程序迭代、数据降维等比较复杂的算法，真正计算起来是非常困难的。因此，基于云计算的海量数据挖掘技术受到了学术界和工业界的共同关注，并且成为现今热点技术中的一员。

1. 大数据特点

（1）大数据来源及数量

提到数据，相信 IT 从业人员首先想到的是数据库、数据仓库等技术，毕竟这是一种至今仍然十分流行且占据主导地位的技术。但请记住，这些技术是构建在关系型数据库理论基础上的，具有明显的结构化特征，换言之，存储在数据库、数据仓库中的数据是通过分析、建模之后筛选出来的自认为有意义的数据。而在这个过程中，已经摒弃掉了许多自认为无意义的数据，真的没有意义吗？答案当然是否定的。但为什么要摒弃呢？原因很简单，以前的技术条件不允许存储如此庞大的数据量。随着物联网概念的提出、应用和发展，每天传感器、控制器、智能设备中都会产生海量数据。据统计，在 2000 年，全球存储了 800 000 PB 的数据；预计到 2020 年，这一数字会达到 35ZB。所以，可以得出这样的结论：大数据从来就是存在的，只是因为技术条件的限制而没有得到重视或是故意规避而已。

（2）大数据类型

以前保存的数据类型主要是结构化数据。然而，并非所有的数据都是可以结构化的，据统计，可结构化数据，即可以存储在数据库等传统系统（主要是指关系型数据库产品） 中的数据占数据总量的 20%左右；其他 80%的数据不能至少是不便于存储于传统的系统中，因为其结构形式是非结构化的或者是半结构化的（如文本、传感器数据、音频、视频、事务及地震模型类的动态数据等非关系型数据）。所以，从这个角度来讲，大数据的类型从结构类型入手可以分为结构化、半结

构化、非结构化数据 3 类。

（3）大数据处理速度及方式

面对如此庞大的数据量以及丰富（至少不再是单一的）的数据，不难想像，对于这些数据的处理速度将会成为企业应用、洞察关键事件的瓶颈。尽管目前还没有得到具体的可度量的值来说明这个问题，但换个角度来考虑，就日常工作中所使用的存储器的存储能力、CPU 频率的变化及不高的工作效率，就完全可以说明数据增长速率对数据处理速度的影响。

建议换个角度来考虑这个问题。随着物联网时代的到来，RFID、传感器等产生的信息流将导致产生大量的传统系统无法处理的持续数据流。请牢记一点，现在处理的是 PB 级的数据流，而非 TB 级的，将来要处理的是 ZB 级甚至有可能更高。所以，需要考虑针对数据产生、流动的速度而进行的数据处理方式的变革，如流数据处理；不再是单纯地处理传统系统中的批量数据。

（4）大数据处理技术及策略

近年来，关于大数据处理技术的探讨一直不断，这方面最具代表性的就是 Hadoop 框架，其本质是一个用于分析大数据集的机制，不一定位于数据存储中，可以扩展到无数个节点，处理所有活动和相关数据存储的协调。Hadoop 方法建立功能到数据的模型，而非传统的数据到功能的模型，这样就可以从可扩展性和分析的角度发现曾经几乎不可能的大数据处理变成可能。由于 Hadoop 部署的复杂性及不稳定性，使其应用到目前为止还不是十分广泛，但无论如何，其为大数据处理提供了一种途径和方式。IBM 在 Hadoop 的基础上发展了 GPFS （General Parallel File System，通用并行文件系统）无共享集群及相关技术，提升了静止大数据处理效率；此外，还提出了 SPL（Streams Processing Language，流处理语言），使得对流数据的处理成为现实并大大提升了实际工作效率。对于大数据的处理策略可作如下理解：

- 按照类型进行分类处理；
- 对分类数据进行分类存储或流处理；
- 对经流处理的非结构化存储部分可转存到传统存储系统，也可直接生成数据应用；
- 对传统存储系统进行批量处理生成数据应用。

2. 基于云计算的海量数据抽取

数据挖掘，也称为数据库中的知识发现过程，就是从海量数据中发现新颖的、有效的或者可能有潜在作用的、最终可被理解的模式的过程。对于企业来说，最终的目的是从海量数据中提取出可理解的知识，并且希望数据规模越大越好，这样挖掘出的知识才更加准确。这么高要求的数据挖掘对开发环境和应用环境也有比较高的要求。在这种情况下，基于云计算的方式是比较适用的。云计算平台中数据中心可以存储海量数据，并可以根据数据挖掘应用的需求对资源进行动态分配，保证数据挖掘算法的可扩展性，并采用容错机制来保证数据挖掘应用的可靠性。

（1）云计算数据抽取优势

- 基于云计算的模式可以进行分布式并行数据挖掘，实现高效实时的挖掘。同时可以适应规模不同的组织，为中小企业带来新型低成本计算环境，大企业云计算平台对某些特定数据

的计算对大型高性能机的依赖性会得到减轻。

- 基于云计算的数据挖掘开发方便，底层被屏蔽掉了。对于用户来说，无需考虑数据的划分、数据分配加载到节点以及计算任务调度等。
- 在并行化条件下利用原先的设备，可以在很大程度上提高大规模处理数据能力。在增加结点方面也比较自由和方便，同时容错性得到了提高。
- 基于云计算的数据挖掘保证了挖掘技术的共享，降低了数据挖掘应用的门槛，使海量数据挖掘需求得到了满足。

（2）基于云计算的海量数据抽取模型

基于云计算的海量数据抽取服务的主要目标是利用云计算的并行处理和海量存储能力，解决数据挖掘面临的海量数据处理问题。基于云计算的海量数据挖掘模型大体上可以分为三层。位于最底层的是云计算服务层，提供分布式并行数据处理及数据的海量存储。云计算环境中对海量数据的存储既要考虑数据的高可用性，又要保证其安全性。云计算采用分布式方式对数据进行存储，为数据保存多份副本的冗余存储方式保证了当数据发生灾难时不影响用户的正常使用。目前常见的云计算数据存储技术有非开源的 GFS（Google File System）和开源的 HDFS（Hadoop Distributed File System），其中 GFS 是由 Google 开发的，HDFS 是由 Hadoop 团队开发的。此外，云计算使用并行工作模式，能够在大量用户同时提出请求时，迅速给予回应并提供服务。

位于云计算服务层之上的是数据挖掘处理层，这一层又包括海量数据预处理和海量数据挖掘算法并行化。海量数据预处理主要是对海量不规则数据事先进行处理。没有好的数据就没有好的数据挖掘结果。由于云计算环境下的 MapReduce 计算模型适用于结构一致的海量数据，因此，面对形态各异的海量数据，首先就要对它们进行预处理。数据预处理方法包括数据抽取、数据转换、数据清洗和集成、数据规约、属性概念分层的自动生成等。经过预处理的数据能提高数据挖掘结果的质量，使挖掘过程更有效、更容易。

海量数据挖掘的关键是数据挖掘算法的并行化。由于云计算采用的是 MapReduce 等新型计算模型，需要对现有的数据挖掘算法和并行化策略进行一定程度的改造，才有可能直接应用在云计算平台上进行海量数据挖掘任务。因此需要在数据挖掘算法的并行化策略上进行更为深入的研究，从而使云计算并行海量数据挖掘算法的高效性得以实现。并行海量数据挖掘算法包括并行关联规则算法、并行分类算法和并行聚类算法，用于分类或预测模型、数据总结、数据聚类、关联规则、序列模式、依赖关系或依赖模型、异常和趋势发现等。基于此，针对海量数据挖掘算法的固有的特点对已经存在的云计算模型进行优化升级以及适当扩充，使其对海量数据挖掘的适用型得到最大程度的提升。

最顶层是面向用户的用户层，该层主要接收用户的请求，将它传递给下面两层，并将最终的数据挖掘结果展示给用户。用户通过友好的可视化界面管理和监视任务的执行，并且可以很方便地查看任务执行结果。用户的数据挖掘请求通过用户输入模块传递到系统内部，系统根据用户提交的一些数据挖掘参数和基本数据，在算法库中选择合适的数据挖掘算法，然后调用经过预处理阶段的数据，分配到 MapReduce 平台上进行并行数据挖掘，挖掘出的结果通过结果展示模块传递给用户。海量数据的存储和并行化处理都依赖于云计算环境。

（3）基于云计算的数据挖掘模型的不足及后续工作开展的方向

由于云计算还处于高速发展时期，必然会面临很多挑战，基于云计算的数据挖掘中也同样存在着一些问题。

- 云计算带来的需求问题。基于云计算的数据挖掘，最终会发展成为一种云服务模式，必然会面临着多样化和个性化的需求。
- 海量数据的问题。从数量上来说，可能需要处理数量级达到 TB 级乃至 PB 级的数据，另外还有高维数据、各种噪声数据以及动态数据等，这都为数据处理带来了极大的困难。
- 算法的选择问题。选择合适的算法及并行策略来完成任务是最关键的问题。另外，算法的设计、参数的调节都会直接影响到最终的结果。
- 不明确性问题。数据挖掘过程中可能会存在许多不明确性，进行数据挖掘的目的就是要将这些不明确性带来的影响降到最低。这些不明确性包括对数据挖掘任务描述的不明确性、进行数据采集和预处理时会出现的不明确性，数据挖掘方法选择和最终结果的不明确性以及对如何评价数据挖掘结果的不明确性等。

针对以上提出的问题，后续工作可以从以下几个方面着手:

- 基础设施建设方面，根据多样化和个性化需求，并综合考虑到各领域各行业的特点，构建专属的数据挖掘云服务平台。
- 虚拟化技术为数据挖掘云服务提供了重要的技术支持，后续应加大对虚拟化技术的研究开发，并促进其成果的广泛应用，高效地对计算资源实现自主分配和调度。
- 在云服务应用产品的研发环节中，应多考虑社会实际需求，并大力引导公众积极参与其中，这样就可以更好地满足数据挖掘个性化、多样化的需求。
- 在可信性方面，使用的算法最好具有通用性，并且可以随时进行检查、调整以及查看。
- 数据安全问题不能像一般的信息安全那样直接加密，应该是由客户根据自己的需求，在自己的平台终端上自主通过适当加密措施对数据进行保护。

未来数据挖掘云服务将会有很好的势头，更多的专业人士会成为服务的供应商，公众和各种企业组织机构会从这项服务中获益良多，数据挖掘研究受计算环境的影响将降低，其应用范围也将大大拓宽。

3.4 数据清洗技术的实现

3.4.1 数据清洗流程

可以将数据清洗的过程分成以下几个阶段。数据清洗过程主要包括数据预处理，确定清洗方法，校验清洗方法，执行清洗工具和数据归档 5 个阶段。每个阶段还可以再细分若干任务。这 5

个阶段可以描述为：

- 数据预处理。在数据清洗的最初阶段，往往是对数据进行预处理，以检查数据源的记录是否存在各种问题，并得出有关特征。这个阶段包括数据元素化（Elementizing）、标准化（Standarding）等。
- 确定清洗方法。根据数据源的特点，确定相应清洗方法。
- 校验清洗方法。在正式执行清洗之前，先要验证所用的方法是否合适。往往是从数据源中抽取小样本进行验证，判断其召回率和准确率，如果没有达到要求，还需要对清洗方法进行改进。
- 执行清洗工具或程序。经过校验的清洗方法，其算法经编程后，得到可执行的清洗程序然后对数据源执行清洗操作。
- 数据归档。数据清洗的执行中和执行后往往还需要人工操作，将新旧数据源分别做归档处理，这样可以更好地进行后续的清洗过程。

数据清洗的原理，就是通过分析“脏数据”的产生原因和存在形式，利用现有的技术手段和方法去清洗“脏数据”，将“脏数据”转化为满足数据质量或应用要求的数据，从而提高数据集的数据质量。数据清洗主要利用回溯的思想，从脏数据产生的源头上开始分析数据，对数据集流经的每一个过程进行分析，从中提取数据清洗的规则和策略。最后在数据集上应用这些规则和策略发现“脏数据”和清洗“脏数据”。这些清洗规则和策略的强度，决定了清洗后数据的质量。一般情况下，数据清洗的基本流程如下。

1. 数据分析

数据分析是数据清洗的前提与基础，通过详尽的数据分析来检测数据中的错误或不一致情况，除了手动检查数据或者数据样本之外，还可以使用分析程序来获得关于数据属性的元数据，从而发现数据集中存在的质量问题。

一般情况下，模式中反映的元数据对于判断一个数据源的数据质量是远远不够的。因此分析具体实例来获得有关数据属性和不寻常模式的元数据就变得很重要。这些元数据可以帮助发现数据质量问题，也有助于发现属性间的依赖关系，根据这些依赖关系实现数据转换的自动化。

数据分析主要有两种方法：数据派生和数据挖掘。数据派生主要对单独的某个属性进行实例分析。数据派生可以得到关于属性的很多信息，比如，数据类型、长度、取值区间、离散值和它们的出现频率、不同值的个数，出现空缺值的次数和典型的字符串模式等。通过对数据应用领域的理解以及应用数理统计技术，可以得到属性值的平均值、中间值、最大值、最小值和标准差等统计值。

数据挖掘帮助在大型数据集中发现特定的数据模式。可以通过数据挖掘来发现属性间的一些完整性约束，如，函数依赖或者一些特定应用的商业规则等。它们可以用来填充缺失值，纠正不正确的值和确定多数据源间的重复记录。比如一个有着很高的置信度的关联规则可以暗示出凡是违背它的数据都可能含有某些数据质量问题，需要进一步的检查。

2. 定义清洗转换规则与工作流

根据上一步进行数据分析得到的结果来定义清洗转换规则与工作流。根据数据源的个数，数据源中不一致数据和“脏数据”多少的程度，需要执行大量的数据转换和清洗步骤。要尽可能地为模式相关的数据清洗和转换指定一种查询和匹配语言，从而使转换代码的自动生成变成可能。

3. 验证

定义的清洗转换规则和工作流的正确性和效率应该进行验证和评估。可以在数据源的数据样本上进行清洗验证，当不满足清洗要求时要对清洗转换规则、工作流或系统参数进行调整和改进。真正的数据清洗过程中往往需要多次迭代地进行分析、设计和验证，直到获得满意的清洗转换规则和工作流。它们的质量决定了数据清洗的效率和质量。

4. 清洗数据中存在的错误

在数据源上执行预先定义好的并且已经得到验证的清洗转换规则和工作流。当直接在源数据上进行清洗时，需要备份源数据，以防需要撤销上一次或几次的清洗操作。清洗时根据脏数据存在形式的不同，执行一系列的转换步骤来解决模式层和实例层的数据质量问题。为处理单数据源问题并且为其与其他数据源的合并做好准备，一般在各个数据源上应该分别进行几种类型的转换，主要包括：

（1）从自由格式的属性字段中抽取值（属性分离）。自由格式的属性一般包含很多信息，而这些信息有时候需要细化成多个属性，从而进一步支持后面重复记录的清洗。

（2）确认和改正。这一步骤处理输入和拼写错误，并尽可能地使其自动化。基于字典查询的拼写检查对于发现拼写错误是很有用的。

（3）标准化。为了使实例匹配和合并变得更方便，应该把属性值转换成一个一致和统一的格式。

5. 干净数据回流

当数据被清洗后，干净的数据应该替换数据源中原来的“脏数据”。这样可以提高原系统的数据质量，还可避免将来再次抽取数据后进行重复的清洗工作。

3.4.2 数据清洗框架

1. 与领域无关的数据清洗框架

元数据是指“关于数据的数据”，指在数据清洗过程中所产生的有关数据源定义、目标定义、转换规则等相关的关键数据。元数据在数据清洗的过程中包括以下几个组件。

（1）基本组件

该功能主要是对元数据的特征进行描述，它包括：可以提供元数据的数据库名，数据库编号，这些数据库的表及表的编号，表中的属性及属性的编号。

（2）清洗规则组件

数据质量规则定义了元数据中的质量问题和数据清洗规则。它包括错误数据表，含有错误类型编号，错误表现形式，可能的修改规则编号等。转化公式表含有数据格式之间的转换公式。同时，这个组件还包括一张数据清洗规则表，含有可能清洗规则的定义等。

（3）数据加载组件

数据加载组件是用于确定异构的元数据什么时候将什么数据加载到目的数据库中。它包括输出模型表，反映了清洗后的数据到目的数据库之间的映射等。

与领域无关的数据清洗框架由 3 个工作流构成。具体清洗过程说明如下：

（1）数据分析工作流

分析所要清洗的数据源，定义出数据清洗的规则，并选择合适的清洗算法，使其能更好地适应所要清洗的数据源。

（2）数据清洗工作流

把数据源中需要清洗的数据通过接口调入到中间数据库中来。调用算法库中的相应算法对数据源进行预处理，如数据标准化，并根据预定义的规则，把数据记录中的相应字段转化成同一格式。然后，分步执行数据清洗，其清洗过程一般为：首先清洗错误数据、然后清洗不完整数据，最后相似重复记录。

（3）清理结果检验工作流

数据清洗运行结束后，在系统窗口中显示出数据清洗结果，根据清洗结果和警告信息，手工清洗不符合系统预定义规则的数据、处理未清洗的数据，从而完成系统的数据清洗。此外，通过查看数据清洗日志，可以检验数据清洗的正确性，对清洗错误进行修正。

2. 基于领域知识的数据清洗框架

与领域知识相关的数据清洗一定要结合应用领域的知识。例如：在数据清洗中利用何种形式来表示领域知识，怎样抽取、验证、优化知识，什么类型的知识适合于数据清洗，如何管理知识，等等。

基于知识的数据清洗框架，在领域知识的指导下从样本数据中抽取、验证知识，然后通过专家系统引擎对整体数据进行清洗，对于系统不能处理的数据，通过用户参与进一步处理。同时，系统可以通过机器学习的方法不断修改和优化规则库，以后碰到类似情况时，它就知道怎样做出相应的处理了。这个框架包含 4 个阶段。

（1）规则生成阶段

在这个阶段，首先要生成一个样本数据集，样本数据集是从整个数据库中抽取出的一小部分样本数据，在此基础上通过专家的参与产生规则库。在得到初步的规则之后，把它们应用到样本数据集上，通过观察中间结果，可以进一步修改已有规则，或者添加新的领域知识，如此反复，直到对所得结果满意为止。在这个过程中，可以用机器学习或者统计学技术来帮助建立规则，降低所需的人工分析工作量。

（2）预处理阶段

在这一阶段，根据生成的预处理规则纠正我们能检测到的所有异常。基本的预处理包含：数据类型检测，数据格式标准化，解决数据中不一致的缩写。

可以用查找表完成这样的转换，转换通常与领域知识有很密切的联系。这一阶段将输出一个满足一定条件的记录集合，而它将作为下一步处理的输入。这个阶段所做的预处理是可扩展的，针对不同的数据清洗种类会有不同的内容。

（3）处理阶段

满足一定条件的预处理后的数据接着流入带有一个规则库的专家系统引擎，典型的规则包括：

- 脏数据检测规则：这些规则指确认脏数据的条件。
- 重复数据检测/合并规则：这些规则指定如何检测/合并重复数据，一个简单的合并规则是在一组重复的记录里面，保留最近使用记录而把其余的记录删除掉。
- 错误数据更正规则：这些规则指定在特定的情况下改正脏数据的方法。当预处理过的数据流入专家系统引擎后，便激发这些规则。规则库是可扩展的，针对不同的业务需求将会包含不同的规则。

规则库中含有系统日志，用来跟踪记录处理阶段所有的操作及其原因，通过检查日志进行一致性和准确性检查，一旦发现错误，还可以撤销错误的数据清洗，还可应用它来检查规则库的有效性，如果一个规则经常错误地归类重复记录，或者错误地修改值，那么就应该删除或修改此规则。

（4）数据加载阶段

通过数据加载规则，把清洗后的数据加载到目的数据库中。在整个数据清洗的体系框架中，无论是元数据库中定义的规则还是规则库中的规则，规则的定义与执行是数据清洗的主线。在清洗框架中，对于清洗规则的执行，既可采用批量执行，也可采用即时执行。批量执行对整体来说执行速度较快，但是即时执行交互性更好，清洗质量一般也较高。这里的数据清洗规则由用户定义，一个比较完整的规则用户需要完整表述清洗范围、检查条件和处理方法。

相似重复记录可以采用自动匹配检测。在程序自动匹配发现相似重复记录时，自动产生规则，填写规则中的部分项。例如在进行相似重复记录的合并时，对于可以自动处理的，规则中可以预先定义处理策略，如两条记录之间没有信息互补关系，表示的信息内容完全一样，这样直接删除其中一条即可，否则，应该交由用户手工处理。

3. 数据清洗框架设计

为了将数据清洗系统面临的复杂的多数据源实例化问题转化为相对比较简单的单数据源实例化问题，我们在数据清洗之前设计了数据预处理部分。在数据清洗的核心数据清洗引擎部分，针对信息集成系统存在的数据质量的实际问题，结合数据清洗的基本原理和分别设计了 4 个模块：数据选取模块、数据标准化模块、重复性判断模块和映射模块。如图 3.3 所示。

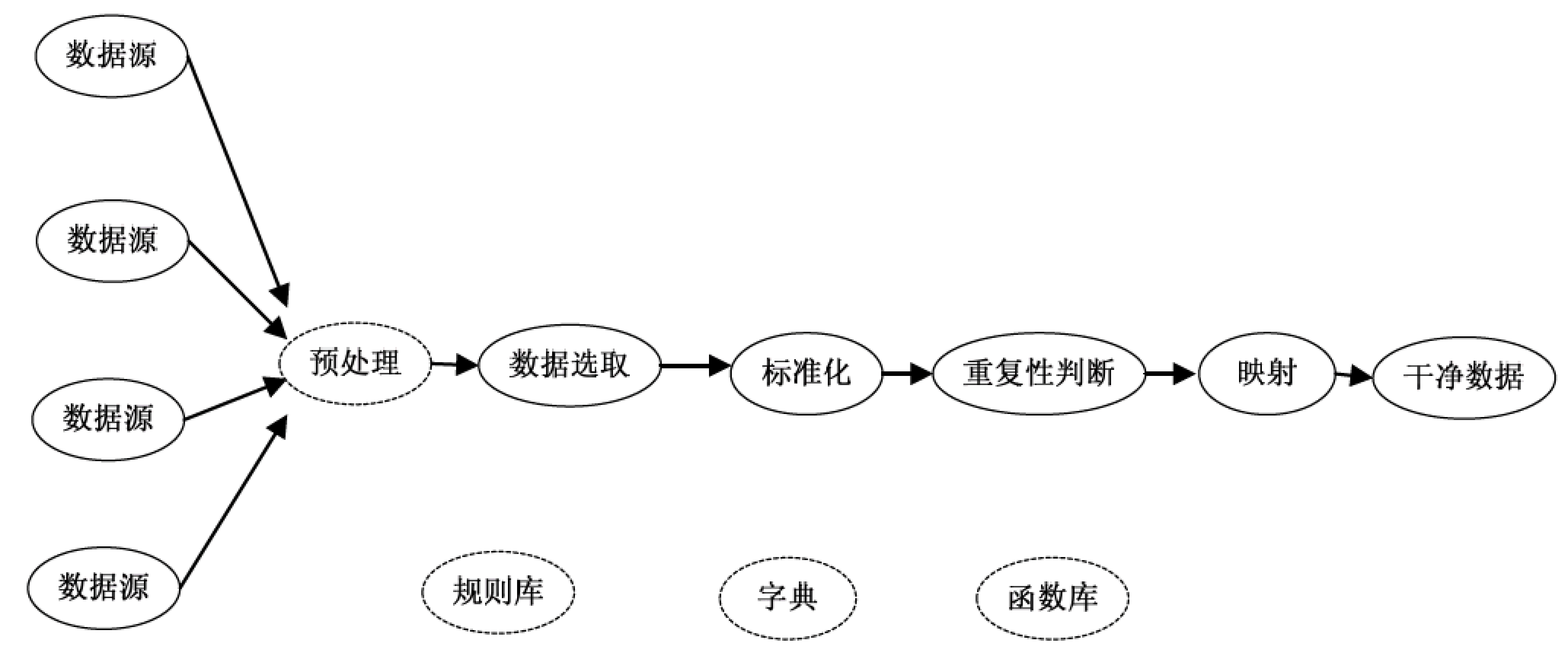


图 3.3　数据清洗流程图

首先，为了提高数据清洗的效率，仅对用户需要的数据进行清洗，而不是对所有数据进行清洗，因此，需要设计一个数据选取模块来对用户需要的数据进行提取；由于选取的数据格式及表达方式不同，给判断其重复性带来不便，在这里设计了数据的标准化模块来解决标准化规则问题；所有的前期准备工作都是为了判断数据的重复性，它是数据清洗引擎的核心部分，为此设计了重复性判断模块；对数据的重复性进行判断后，为了不影响原始数据，将对原始数据采取映射操作，将其组织成新的干净数据。

整个数据清洗框架由数据预处理、数据清洗引擎、系统维护及扩展接口三个部分组成。

（1）数据预处理

数据预处理的主要作用是将多数据源提供的 XML 文档进行简单的数据整合，把数据质量问题由原来比较复杂的多数据源的实例问题转化为单数据源实例问题。

（2）数据清洗引擎

数据清洗引擎是数据清洗系统的核心部分，主要由数据选取模块、标准化模块、重复性判断模块和映射模块 4 个部分组成。

- 数据选取模块

数据预处理结束后，本系统面对的问题将由原来的多个脏文档转变为一个脏文档，在本模块中将对预处理部分输出的脏文档数据进行选取，获取脏数据中用户需要的数据作为待清洗的数据，以提高数据清洗的清洗效率。

- 标准化模块

该模块负责将选取数据的格式、表达方式统一化，系统维护及扩展接口模块中的规则库提供了标准化的规则。

- 重复性判断模块

主要功能是将标准化后的数据进行笛卡尔积、匹配和聚类等操作以判断哪些元素（属性）是

重复性元素（属性）。系统维护及扩展接口模块中提供的规则库和字典将参与重复性判断过程：重复性判断主要依赖于规则库提供的相应判断规则；判断过程中语义问题的解决则主要依赖于对字典的查询。

- 映射模块

为了保持脏数据的原始性，将不对原始数据进行修改，而仅是运用重复性判断模块中提供的结果（哪些元素或属性是重复的）以及用户选取模块获得的数据集合，将数据集合中的数据进行一系列映射，从而获得干净数据。

（3）系统维护及扩展接口

为了保证本数据清洗系统的可维护性和可扩展性，提供了系统维护及扩展接口部分，该部分主要由三个模块组成。

- 规则库

主要负责提供对数据标准化的规则以及数据重复性判断的相应规则，用户可以根据具体的数据清洗对该规则库进行相应的添加、删除和修改等操作。

- 字典

主要是针对在数据进行重复性判断过程中出现的简单语义问题和原始数据的输入错误等问题，提供元素（属性）重复性判断的依据。

- 函数库

将开发本系统的函数统一放置到该函数库中，以备对本系统的进一步升级和维护。

3.4.3　数据清洗相关技术

1. 不完整数据

在异构数据集成过程中，由于数据源模式的不同以及数据抽取方式的不同或人为的原因等，造成所得到的数据通常并不完整的。数据不完整是产生数据质量问题的一个重要因素，在这里引入一个不完整数据的定义。为了清理不完整数据，一般采取两个步骤来完成，首先检测不完整的数据，其次对不完整数据的处理，对不完整数据的处理又可分成以下三步。

（1）判断数据的可用性

如果一条记录中字段值缺失得太多，或者是关键的字段值缺失，就没有必要去处理该记录。因此，对于检测出的不完整数据，要根据每一条记录的不完整程度以及其他因素，来决定这些记录是保留还是删除。判断数据的可用性就是完成这一工作。

（2）忽略缺失字段的值

对于不重要的字段值缺失的方法，一般是采取删除属性或记录的方法。

（3）填充缺失字段的值

填充缺失字段的值是指对那些要保留的记录，要采取一定的方法来处理该记录中缺失的字段值，然后删除不可用的记录。在多数情况下，数据源之间的字段值并不是相互独立的。所以通过字段值之间的关系可以推断出缺失的字段值，然后填充所缺失的字段值。

使用忽略缺失字段值的清洗方法，比较简单，但也有可能将潜在的有价值的信息一并删除，这比含有不完整数据的情况还要严重。因此一般建议是把那些不完整的数据填充，而不是删除掉。缺失值填充算法也是数据清洗研究的热点之一。缺失值的填充，即把缺失值用最接近它的值来替代，从而提高可用数据的质量。

对于不完整数据，一般采取以下几种处理方法。

（1）常量值替代法

常量替代法就是对所有缺失的字段值用同一个常量来填充，采用的常量可以为数据集的最大值或者最小值，由于所有的缺失值都被当成同一个值，容易导致错误的结果。

（2）采用统计的方法

这类方法主要通过对数据的分析，得出数据集的统计信息，然后利用这些信息填充缺失值。其中最简单也最常用的方法是平均值填充方法和最大概率填充方法。均值填充法是最常用的缺失值填充法，它把完整数据的算术平均值作为缺失数据的值。它根据的是正态分布的原理，“在正态分布下，样本均值是估算出的最佳的可能取值。”应用均值填充法将会影响缺失数据与其他数据之间的相关性。最大概率法是选择数据集中出现次数最多的值来填充缺失值。

（3）采用估算值的方法

估算值替代法是比较复杂，但也是比较科学的一种方法。采用这种方法来填充缺失字段值的过程为：首先采用相关算法，如判定树归纳等算法预测该字段缺失值的可能值，然后用预测值填充缺失值。

（4）采用分类的方法

分类的概念是在已有数据的基础上构造出一个分类函数或模型，即通常所说的分类器（Classifier）。该函数或模型能够把数据库中的数据记录映射到给定类别中的某一个类别。数据分类技术，如贝叶斯网络、神经网络、粗糙集理论等也都用来对缺失值处理。

2. 异常数据处理

异常数据的产生可能有多种原因：数据源本身难以得到精确的数据，收集数据的设备可能出现故障，在数据输入时可能出现错误，数据传输过程中可能出现错误，存储介质有可能出现损坏等。数据错误是最重要的数据质量问题。简单地说，数据错误是指数据源中记录字段的值和实际的值不相符。

在数据清洗中异常数据的处理是数据清洗的一个重要环节。在对含有异常数据进行清洗的过程中，现有的方法通常是找到这些含有异常数据的记录并删除掉，其缺点是事实上通常只有一个属性上的数据需要删除或修正，将整条记录删除将丢失大量有用的、干净的信息。在异常数据的数据清洗中，许多文献提出了噪声数据的概念，下面给一个简单的定义：噪声数据是指包含错误的数据或存在偏离期望的孤立点值。

噪声数据的处理主要分为 3 个步骤：

- 识别噪声数据，并判断是否可以判定引起噪声的属性。
- 对于能判定引起噪声的属性的记录，用干净数据（包括清洗过的噪声数据）包含的信息对其进行矫正；对于不能判定引起噪声的属性记录，根据“噪声记录去除非噪声属性后的仍然是噪声记录”这个基本原则，判定其引起噪声的属性，并进行矫正。
- 在矫正过程中生成噪声在属性上的分布统计。

噪声数据处理的方法有：

- 分箱（Binning）：利用属性值的相邻性进行数据的平滑化。将这组属性值按照大小次序排成一个线性队列，再按照一定的步长将其分成若干个小组，最后就每个小组局部进行数据的平滑化。
- 聚类（Clustering）：将一组数据按照某种相似性划分为若干组，如数据值的大小、数据语义的分类等，而那些遗留在分组之外的零散数据将被作为一种噪声数据而剔除。
- 人机结合检查：可以通过人工检查和计算机结合的办法来识别孤立点。
- 回归（Regression）：定义一个回归函数来平滑数据。线性回归涉及找出适合两个变量“最佳”直线，使得一个变量能够预测另一个，多线性回归是线性回归的扩展，它涉及多个变量，数据要适合一个多维面。

3. 重复记录处理

产生重复记录的原因有很多，包括数据录入不正确、数据本身不完整、数据的演变、数据缩写及拼写错误等，这些因素使得数据源中存在大量不一致的、重复的数据。因此，准确高效地识别数据源中的重复数据，消除矛盾的数据，被认为是数据清洗最主要问题之一。

在数据集成过程中，由于不同数据库之间对数据表示的差异或者因为人为的差异导致集成后的数据库中同一实体对应多条记录，这些重复的记录可能导致建立错误的数据挖掘模型，给后期数据的决策分析产生很大的影响。因此，判断两条记录是否相似重复在数据集成、数据挖掘中尤为重要。

所谓相似重复记录是指客观上表示现实世界中的同一实体，但由于表述方式不同或因其他原因而使数据库不能识别其为重复的记录。在关系数据库中，如果两条记录在所有的属性上的值都完全相同，就可认为是重复的。

相似重复记录判断是一个复杂的问题。在关系数据库中判断两条记录是否重复，这需要通过记录的比较决定记录间的相似程度，即通过记录各字段值语法上的比较结果，决定两条记录语义上的等价性，这也称为记录的匹配问题。现实中的数据又是比较复杂的，两条记录是否同一实体

有时还要根据实际情况来判断。

要想清理数据源中的相似重复记录，必须要先通过某种方法检测出相似重复记录，然后采取一定的策略清除这些重复记录，目前比较常用的重复记录清洗是先将数据库中的记录排序，然后，通过比较邻近记录是否匹配来检测相似重复记录。

重复记录清洗的基本过程一般包括以下几个阶段。

（1）记录排序

- 预处理：制定初步的记录匹配策略，建立算法库和规则库。
- 初步聚类：主要是对数据库中的记录进行初步排序。

（2）相似记录检测

- 字段匹配。选择用于记录匹配的属性，调用算法库中字段匹配算法，计算出字段的相似度。
- 记录匹配。根据属性在决定两条记录相似性中重要程度的不同，为每个属性分配不同的权重，调用算法库中记录匹配算法，根据上一步中字段相似度的结果计算出记录相似度，判断是否是相似重复记录。
- 重复记录检测。在数据库应用检测重复记录的算法对整个数据集中的重复记录进行检测。为了能检测到更多的重复记录，一次排序不够，要采用多轮排序，多轮比较，每次排序采用不同的键，然后把检测到的所有重复记录聚类到一起，从而完成重复记录的检测。

（3）相似记录合并/清除

根据已定义的规则库中的合并/删除规则，对同一重复记录聚类中的重复记录进行合并或者删除，只保留其中正确的那条记录。重复记录清洗的完整性和准确性是很重要的。例如在银行管理系统中，如果没有对相同客户记录进行匹配，银行会把一个客户当作两个甚至更多客户对待，客户数量就被夸大了，根据美国 Meta 集团的研究，银行客户资料约有 5%~12%是重复的。另一方面，如果把本不应该合并的记录合并了，这时对客户的看法也是错误的。这些不完整、不准确和不可靠的重复记录都会导致不准确的分析结果和决策，导致银行费用增加和利润减少：如对客户信用等级有着错误的认识，就可能导致投资风险；对某个客户价值没有充分认识，就可能导致失去顾客；错误记录可能导致营销资源的浪费。

4. 异常记录检测的常用算法

数据清洗的一个关键技术是异常记录检测，下面主要阐述一些常用的异常记录检测算法。

（1）统计学算法

统计学算法是基于模型的算法，即为数据创建一个模型，并且根据对象拟合模型的情况来评估它们。其中回归分析是应用极其广泛的数据分析方法之一，它提供了一套描述和分析变量间相关关系，揭示变量间的内在规律，并用于预测、控制等问题的行之有效的方法。在现实世界中，许多变量（或通过适当变换的变量）之间都有或近似具有线性相关关系，又因为线性回归分析方法简单、理论完整，因此线性回归模型常常作为数据分析的首选模型。线性回归模型可由如下公

式描述：一般说来，同一问题所涉及的众多变量之间会存在一定的相关性，这种相关性会使各变量的信息有所“重叠”，于是人们希望对这些彼此相关的变量加以“改造”，用为数较少的、信息互不重叠的新变量来反映原变量提供的大部分信息，从而通过对为数较少的新变量的分析达到解决的目的。主成分分析和典型相关分析便是在这种降维的思想下产生的处理高维数据的统计方法。

（2）聚类算法

聚类分析是依据样本间度量标准将其自动分成几个组群，且使同一群组内的样本相似，而属于不同群组的样本相异的一组方法。一个聚类分析系统的输入是一组样本和一个度量样本间相似度（或相异度）的标准。聚类分析的输出是数据集的几个组，这些组构成一个分区或一个分区结构。聚类的样本是用度量指标的一个向量表示，即用多维空间的一个点来表示。下面介绍几个比较经典的聚类算法。

- 层次聚类。大多数层次聚类过程不是基于最优的思想，而是通过反复的分区直至收敛，找出一些接近最优标准的解决方案。层次聚类算法分两类：分裂算法和凝聚算法。分裂算法从整个样本集 X 开始，把它分成几个子集，然后把每个子集分成更小的集合，依次下去。最终，分裂算法生成一个由粗略到精细的分区序列。凝聚算法首先把每一个对象当作一个初始类，然后把这些类合并成一个更粗略的分区，反复合并，最后所有的对象都在一个大类内。一般来讲，凝聚算法更有实际应用价值。
- 分区聚类。分区方法通常利用一个局部定义（在样本子集上定义）或全局定义（在整个样本集上定义）的准则函数进行优化来生成类。一个全局准则，如欧氏平方误差度量标准，再用一个原型或重心表示每个类，然后依据最相似的原理将样本分配给各个类。一个局部的标准，如互邻近距离（Mutual Neighbor Distance，MND）利用数据的局部结构或环境生成类。识别数据空间的高密度区域是生成类的一个最基本准则，最常用的分区聚类方法是基于方差标准的方法。

5. 重复记录检测的常用算法

数据清洗的另一个关键技术是重复记录检测，最可靠的重复记录检测方法是比较数据仓库中每对记录，但该算法时间复杂度太大，需要 N(N-1)/2 次比较，

其中 N 是数据仓库中记录的总数。排序合并算法是检测数据库中重复记录的标准算法，它的基本思想是，先对数据集进行排序，然后比较相邻记录是否相似，这一算法也为在整个数据库级上检测重复记录提供了思路，目前已有的检测重复记录的算法也大多以此思想为基础。在下面的部分将给出几种主要检测重复记录的算法。

（1）基本近邻排序算法 SNM

基本近邻排序算法的基本步骤如下：

- 创建排序关键字。抽取记录属性的一个子集序列或属性值的子串，根据这些子集序列或属性值的子串，计算数据集中每一条记录键值。
- 排序。根据排序关键字对整个数据集进行排序，尽量把潜在的、可能的重复记录调整到邻

近的区域内，便于进行记录匹配。

- 合并。在排序后的数据集上滑动一个固定大小的窗口，窗口可容纳 w 条记录，则每条新进入窗口的记录都要与先前进入窗口的 w-1 条记录进行比较，如检测到重复记录，则进行合并，否则最先进入窗口内的记录滑出窗口，最后一条记录的下一条记录移入窗口，再进行下一轮比较，直到数据集的最后记录移入窗口后比较完毕。

（2）多越近邻排序算法 MPN

该算法的基本思想：独立地执行多趟 SNM 算法，每趟创建不同的排序关键字和使用相对较小的滑动窗口。然后采用基于规则的知识库来生成一个等价原理，作为合并记录的判别标准，将每趟扫描识别出的重复记录合并为一组，在合并时假定记录的重复具有传递性，即计算其传递闭包（transitive closure），所谓传递闭包，是指若记录 R1 与 R2 互为重复记录，R2 与 R3 互为重复记录，则 R1 与 R3 互为重复记录。通过计算每趟扫描识别出的重复记录的传递闭包，可以得到较完全的重复记录集合，能部分解决重复记录漏查问题。

（3）优先权队列算法

优先队列策略借用邻近排序算法的思想，具体策略：先抽取一个或多个字段构成关键字，根据关键字，对数据集进行排序，然后在一个长度固定的子集队列中检测匹配记录，采用类似 LRU 算法（最近最少使用算法）来控制队列的长度。通过匹配操作找出需要合并的子集，计算其传递闭包，然后合并它们，最后得到若干个近似重复记录集。基于这样一个观察：一个简单的关键字在排序后不能完全将重复记录聚集在一起，因此一趟优先队列算法可能遗漏一些重复记录，为避免这种情况，提出多趟优先队列算法，每次采用不同的关键字进行排序。

但是上述算法存在以下缺陷：

- 对排序关键字的依赖性太大。上述算法检测重复记录的精度在很大程度上依赖于所创建的排序关键字，它直接影响着匹配的效率与精度，如果选取关键字不当，可能会遗漏很多重复记录，比如在排序后，有些重复记录的物理位置相距较远，没有同时位于同一个滑动窗口内，因此不能被识别为重复记录。
- 滑动窗口的大小 w 和队列长度的选取很难控制。它们较大时进行的比较次数多，而有些记录之间比较是没有必要的，它们较小时可能会漏配。

6. 数据清洗工具

从特定功能的清洗工具、ETL 工具以及其他工具 3 个方面来对数据清洗工具进行介绍。

（1）特定功能的清洗工具

特定的清洗工具主要处理特殊的领域问题，基本上是姓名和地址数据的清洗，或者消除重复。转换是由预先定义的规则库或者和用户交互来完成的。在特殊领域的清洗中，姓名和地址在很多数据库中都有记录而且有很大的基数。特定的清洗工具提供抽取和转换姓名及地址信息到标准元素的功能，并基于清洗过的数据工具来确认街道名称、城市和邮政编码。特殊领域的清洗工具有 IDCENTRIC、PURE INTEGRATE、QUICKADDRESS、REUNION、TRILLIUM 等。消除重复的

工具根据匹配的要求探测和去除数据集中相似重复记录。有些工具还允许用户指定匹配的规则。目前已有的用于消除重复记录的清洗工具有 DATACLEANSER、MERGE/PURGE LIBRARY、MATCH IT、ASTERMERGE 等。

（2）ETL 工具

现有大量的工具支持数据仓库的 ETL 处理，如 COPYMANAGER、DATASTAGE、EXTRACT、WERMART 等。它们使用建立在 DBMS 上的知识库以统一的方式来管理所有关于数据源、目标模式、映射、教本程序等的原数据。模式和数据通过本地文件和 DBMS 网关、ODBC 等标准接口从操作型数据源收取数据。这些工具提供规则语言和预定义的转换函数库来指定映射步骤。ETL 工具很少内置数据清洗的功能，但是允许用户通过 API 指定清洗功能。通常这些工具没有用数据分析来支持自动探测错误数据和数据不一致。然而，用户可以通过维护原数据和运用集合函数（Sum、Count、Min、Max 等）决定内容的特征等办法来完成这些工作。这些工具提供的转换工具库包含了许多数据转换和清洗所需的函数，例如数据类转变、字符串函数、数学/科学和统计的函数等。规则语言包含 If- then 和 Case 结构来处理例外情况，例如，错误拼写、缩写，丢失或者含糊的值和超出范围的值。而在我国，对数据清洗的研究甚少，还没有一个成型的完善的 ETL 工具应用于数据仓库的系统中。

（3）其他工具

其他与数据清洗相关的工具包括基于引擎的工具（COPYMANAGER、DECIS IONBASE、POWERMART、DATASTAGE、WAREHOUSEADM IN ISTRATOR）、数据分析工具（MIGRATIONARCHITECT、WIZRULE、DATAM ININGSUITE）和业务流程再设计工具（INTEGRITY）、数据轮廓分析工具（如 MigrationArchitect、Cevoke Software 等）、数据挖掘工具（如 WIZRULE 等）。

3.4.4　基于 Hadoop 的数据清洗方案

1. 相关技术简介

（1）Hadoop 简介

Hadoop 是由 Apache 基金会开发，并由其开源组织的一个分布式系统基础架构，可以在大量廉价的硬件设备组成的集群上运行应用程序，为应用程序提供一组稳定可靠的接口，同时用户在充分利用集群的威力高速运算和存储来开发分布式程序，而不了解分布式底层细节。Hadoop 是一个能够对大量数据进行分布式处理的软件框架。其保证了处理的可靠性、高效性、可伸缩性。Hadoop 是可靠的，因为它假设计算元素和存储会失败，为此它维护了多个工作数据副本，以保证能够针对失败的节点重新分布处理。Hadoop 是高效的，因为它的工作运行方式是并行的，可以通过并行处理以加快数据的处理速度。Hadoop 还是可伸缩的，对于 PB 级数据也能够进行处理。除此之外，Hadoop 还可以运行于普通的 PC 机器上，对硬件要求不高，故成本低，任何人都可以使用。Hadoop 的核心是 HDFS 分布式文件系统、Map/Reduce 分布式并行计算框架。通过 HDFS 提

供数据存储，使用 Map/Reduce 实现并行数据处理。同时，Hadoop 的价值还体现在基于这项技术的组件添加、交叉集成和定制实现上。

Hadoop Distributed File System（HDFS）被设计为适合运行在通用硬件上的分布式文件系统。它与现有的分布式文件系统有很多共同点。当然，也存在与其他分布式文件系统的明显区别。HDFS 是一个高度容错性的系统，只需要廉价的机器就能加以部署。同时，HDFS 以其高吞吐量的数据访问，使得它非常适合大规模数据集上的应用。HDFS 是 Hadoop 平台的核心，为分布式计算存储提供了底层支持。

（2）HDFS 目标及假设

- 节点失效

节点失效是正常的，并非突发事件。HDFS 集群实例通常由许多服务器组成，其相应的处理数据也分布在各个机器上。现实应用中的集群系统节点数据是非常多的，而且某个节点或某部分组件失效的概率也非常大。所以快速的硬件错误检测以及及时自动的回复是 HDFS 架构的重要参考指标。

- 海量数据集支持

HDFS 应用程序大部分都需要处理很大的数据集。在 HDFS 文件系统中，一个文件的大小一般都是几 GB 甚至几 TB。HDFS 可以用来优化大文件存储并且能提供集中式的很高的数据带宽，还能够使单个集群支持成百上千个节点。通常在独立的 Hadoop 文件系统中能够支持上千万个文件。

- 数据流式访问

在 HDFS 上的应用程序需要对数据进行流式访问，这些应用程序与普通的应用不一样。HDFS 的关键是提供高数据吞吐量而非数据访问低延迟，其主要应用于数据批处理而非交互式应用。POSIX 标准中的许多约束对于运行在 HDFS 上的应用程序来说是不必要的。可以修改 POSIX 的一些关键性语义以获得在 HDFS 文件系统上的高数据吞吐率。

- 一致性模型

Hadoop 文件系统的应用程序设计成一次性写随机读的文件访问模型。文件一经创建、写入、关闭以后就不会做什么改动。这样简化了数据一致性问题并且保证了数据访问的高吞吐量。

- 异构软硬件平台兼容性

HDFS 文件系统在设计的时候就已经考虑其跨平台性，可以简单的将应用程序从一个平台移植到另一个平台，使得不同应用平台的开发者可以很容易的开发出分布式应用程序。

（3）HDFS 的系统架构

HDFS 文件系统采用的是 master/slave 架构。通常来说，一个 HDFS 集群一般由一个 Namenode 和若干个 Datanode 组成。Namenode 是中心服务器，主要负责管理文件系统的 namespace 和客户端对文件的访问。而 Datanode 负责管理节点上附带的存储，在集群中一般是一个节点一个。在内部，一个文件其实分成一个或多个 Block，并存储于 Datanode 集合中。Namenode 执行文件系统的 namespace 操作，比如打开、关闭、重命名文件和目录等操作，与此同时，Namenode 节点也决

定 Block 到具体 Datanode 节点的映射。Datanode 在 Namenode 的控制下创建、删除、复制 Block。

（4）HDFS 数据存储的保证措施

HDFS 被设计运行在普通硬件上，普通硬件上发生故障是很正常的，所以错误检测并且快速自动恢复是 HDFS 的核心设计目标。为了保障数据存储的进行，HDFS 采取了以下几个措施。

● 冗余备份和副本存放

文件存储时被 HDFS 分成一组数据块，所有的数据块都具有副本。Hadoop 在与客户端相同的节点上放置第一个副本，第二个副本放置在与第一个不同的随机选择的机架上，第三个副本放置在与第二个相同机架上，但是放在不同的节点上。其他副本则放置在随机节点上，并保证相同机架避免放置太多副本。

● 心跳检测机制

集群中的 Datanode 定期向 Namenode 传送数据报告以检测 Datanode 是否处于正常工作状态，这些数据报告被称为心跳包和块报告，出现故障的 Datanode 不会发送心跳包和块报告。Namenode 在接到心跳包后会进行标记，没有发送心跳包的节点被标记为死机，Namenode 不会再向该 Datanode 发送任何请求指令。鼓掌节点会使得副本数量降低，检测到一个副本数量低于定值时重新建立副本。

● 数据完整性检测

HDFS 对文件的数据完整性检测是指在文件创建时记录每个文件数据块的校验和，客户端读取文件时同样会检测该文件数据块的校验和，并将校验和与存储在命名空间的原始校验进行对比，若对比不同，则认为数据块损坏，选择其他的副本进行读取。

● 空间回收策略

HDFS 把被删除的文件放在//trash 目录中，并设定文件在该目录中的存放时间，在未超过存放时间的期限内可以方便地进行文件恢复操作。

● 安全模式和快照

安全模式是指系统启动时 Namenode 的一种特殊状态，此时不进行数据块的复制。Namenode 节点在确认数据库达到最小副本数值或是在允许的范围内会自动退出安全模式状态。HDFS 还可以在数据损坏时利用快照功能回到存储的某一个正确的状态，快照是指对某个时间的数据存储。

（5）MapReduce 简介

MapReduce 是一种编程模型，用于大规模数据集（大于 1TB）的并行运算。概念“Map（映射）”和“Reduce（化简）”，及它们的主要思想，都是从函数式编程语言里借来的，还有从矢量编程语言里借来的特性。它极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。当前的软件实现是指定一个 Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的 Reduce（化简）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

MapReduce 通过把对数据集的大规模操作分发给网络上的每个节点实现可靠性；每个节点会

周期性地把完成的工作和状态的更新报告回来。如果一个节点保持沉默超过一个预设的时间间隔，主节点记录这个节点状态为死亡，并把分配给这个节点的数据发到别的节点。每个操作使用命名文件的原子操作以确保不会发生并行线程间的冲突；当文件被改名的时候，系统可能会把它们复制到任务名以外的另一个名字上去。

Hadoop 的 Map/Reduce 集群主要由两类服务器构成，即作业服务器及任务服务器。其中，作业服务器，也称为 Job Tracker，主要负责管理运行在该架构下的所有作业，也是为各个作业分配任务的核心。其与 HDFS 的 Namenode 类似，也是作为单点存在的，以简化负责的同步流程。而具体负责执行用户定义操作的，是任务服务器 TaskTracker，每个作业被拆分为很多任务，包括 Map 任务和 Reduce 任务等，而作为基本执行的基本单位——任务，它们被分配到合适的任务服务器去执行，任务服务器一边执行一边向作业服务器汇报各个任务的状态，以此来帮助作业服务器了解作业执行的整体情况，分配新的任务等。Map/Reduce 的工作流程主要有 4 个，即提交作业、作业初始化、任务分配、任务执行。

（6）Hive 简介

Hive 是一个构建在 Hadoop 上的数据仓库框架，是 Facebook 2008 年 8 月开源的一个基于 Hadoop 的数据仓库框架，是应 Facebook 每天产生的海量新兴社会网络数据进行管理和机器学习的需求而产生和发展的。其设计目的是让精通 SQL 技能的分析师能够在 HDFS 的大规模数据集中进行查询分析，作为一个通用的、可伸缩的数据处理平台。

Hive 可以将结构化的数据文件映射为一张数据库表，并提供完整的 SQL 查询功能，可以将 SQL 语句转换为 MapReduce 任务运行。其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析。

通过 Hive，可以方便地进行 ETL 的工作。Hive 定义了一个类似于 SQL 的查询语言 HQL，能够将用户编写的查询转换为相应的 Mapreduce 程序基于 Hadoop 执行。其有更丰富的类型系统、更类似 SQL 的查询语言、Table/Partition 元数据的持久化等。故 Hive 更适合于数据仓库的任务，Hive 主要用于静态的结构以及需要经常分析的工作。Hive 与 SQL 的相似性促使其成为 Hadoop 与其他 BI 工具结合的理想交集。

（7）Sqoop 简介

Hadoop 平台的最大优势在于它支持使用不同形式的数据。HDFS 能够可靠地存储日志和来自不同渠道的其他数据。但是为了能够和 HDFS 之外的数据存储库进行交互，MapReduce 程序需要使用外部 API 来访问数据。通常，一个组织中有价值的数据都存储在关系型数据库系统中。

在这种背景之下，Sqoop 应景而生。Sqoop 是一个开源工具，用来将 Hadoop 和关系型数据库中的数据相互转移。它允许用户将数据从关系型数据库（例如：MySQL、Oracle、Postgresql 等）抽取到 Hadoop 中，用于进一步处理。抽取出的数据可以被 MapReduce 程序使用，也可以被其他类似于 Hive 的工具使用。一旦形成分析结果，Sqoop 便可以将这些结果重新导回数据库，供其他客户端使用。

2. 基于 Hadoop 的分布式数据清洗方案

当前，数据清洗面临的主要挑战如下。

- 随着业务的增长，企业的用户数据、日志数据越来越多，数据的错误率也相应增多，使得展现正确数据的能力远远低于用户的需求，故必须提高数据清洗的速度和效率。
- 传统的数据清洗算法在面对海量数据时表现为性能较低、计算能力不如人意，在扩展性、强壮性、伸缩性等方面也较差，这使得传统的数据清洗系统无法扩展其自身的大数据集清洗能力。
- 对应多种数据源的多样的数据格式，在合并清洗的过程中需要保证数据格式的统一性，从而达到很好的处理能力。

海量的数据处理能力和异构数据的兼容处理能力，是基于 Hadoop 分布式数据清洗方案必须解决的问题。利用 Hadoop 生态圈的相关技术及本身核心的集群特性、强大的存储能力及计算能力、灵活的扩展伸缩性，可以很好地解决以上提出的挑战问题。

基本的设计思想是：对于海量数据清洗过程中需要巨大计算能力的各个模块的计算和存储扩展到 Hadoop 集群的各个节点，充分利用 Hadoop 集群强大的计算、存储能力来进行海量数据清洗工作，提高数据清洗的并行性和准确性。对此，采用分层的设计思想，在底层，通过 Hadoop 作为数据格式统一的存储平台，将各种异构数据源的数据统一到 Hadoop 的存储系统当中，并用 Hadoop 来分析处理巨大的待清洗数据；在 Hadoop 层之上，则为相应的并行核心清洗模块，包括数据加载模块、分布式孤立点挖掘模块、结果分析及存储模块，透明地调用 Hadoop 底层的计算和存储能力。概括来说，主要包括以下两点。

（1）存储

在整个方案中，可以使用 HDFS 来存储文件和原始数据。HDFS 高数据吞吐量、强容错性等特点，都可以为海量数据清洗提供效果保证。HDFS 提供了多种访问接口，可以通过 API 和操作命令简单地进行文件或数据的查看。基于 HDFS 文件系统，我们可以将海量的待清洗数据存入，作为一个数据源服务器，将异构数据源的各种数据导入，为数据清洗引擎提供数据输入及相应的结果输出。这里需要注意的是，面对异构的数据源，其多样的数据格式对数据清洗来说，是一个比较大的问题。基于此，这里使用 Hive 数据仓库而不是直接使用 HDFS，来转存异构数据源的数据。对于多种数据源的数据，比如数据库，可以使用 Sqoop 工具直接将其需要清洗的数据导入 Hive 数据仓库，不仅将其存储为结构化的数据文件，同时也将其映射为一张数据表，提供完整的 SQL 查询功能，通过类 SQL 语句快速实现简单的数据预处理任务。Hive 是建立在 HDFS 文件系统上的，故也继承了 HDFS 的各种优点，数据处理结果可以存入 Hive 或 HDFS 中，并通过 Sqoop 写回关系数据库。

（2）计算

对于 Hadoop 平台，其另外一个重要的部分是 MapReduce 运行机制。使用 MapReduce，一方面将各个模块的数据交互联系起来，另一方面也可以将各个模块的计算任务发布到 Hadoop 集群中的各个计算节点。MapReduce 具有很好的扩展性和伸缩性，它屏蔽了下层的具体运行机制，直

接抽象出相应的 MapReduce 等编程接口供快速实现各种算法的并行化。在并行化的过程中，有时需要对多个 Job 进行运算控制，直接使用前一个 Job 的输出作为后一个 Job 的输入。对于无法直接作为输入的 Job，则将该 Job 的输出按照指定格式定位到 HDFS 文件系统，等到下一个 Job 需要使用时才直接从文件进行读取。

3. 方案设计

如图 3.4 所示，Hadoop 数据清洗主要分为 3 个功能模块。

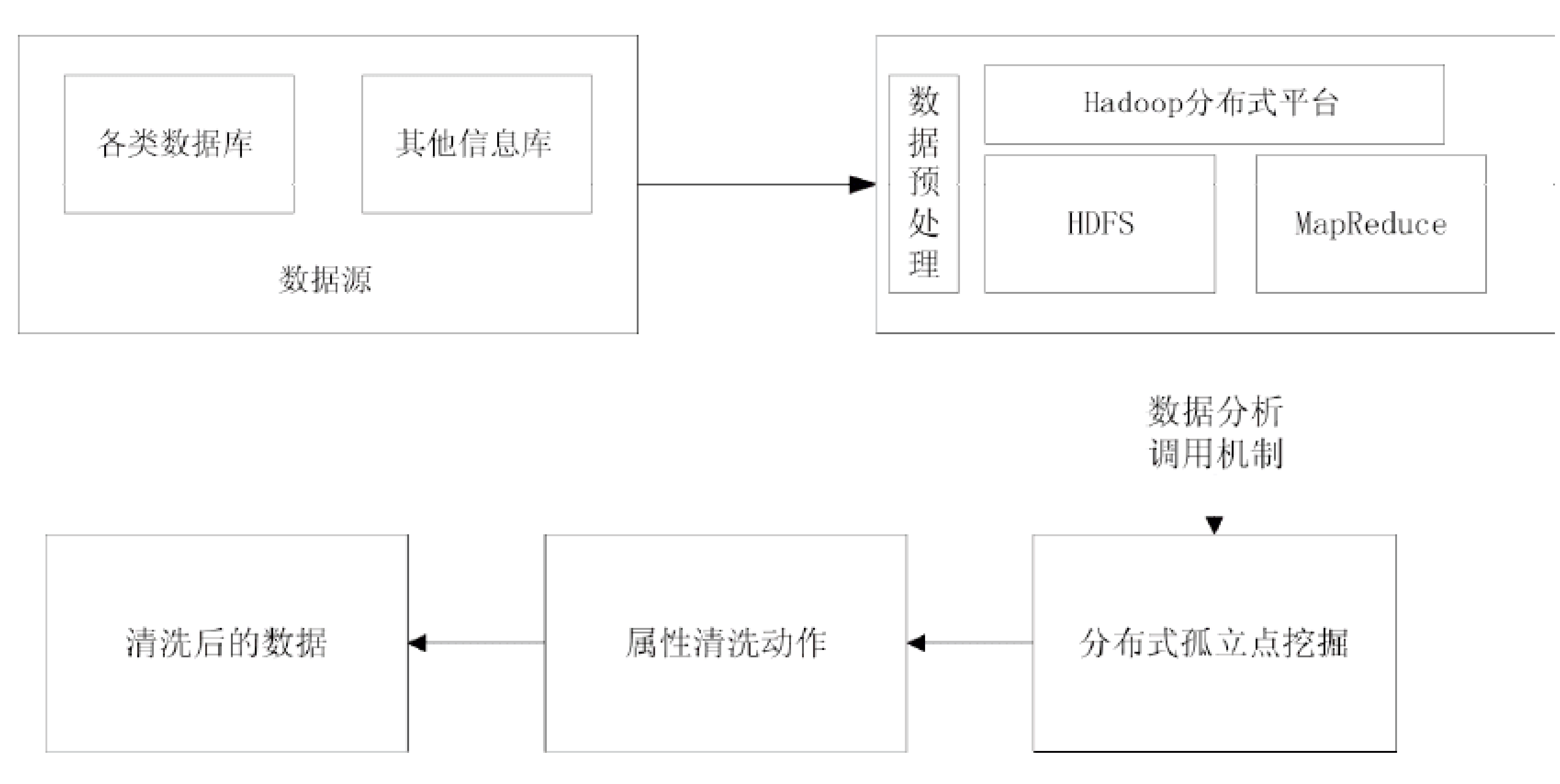


图 3.4　Hadoop 数据清洗流程框图

（1）多源异构数据的装载预处理

Hadoop 可以从任意多的数据源吞入任何类型的数据，可以是结构化数据，也可以是非结构化数据，来自多个数据源的数据可以按任何所需的方式进行合并或聚合，从而实现任意一个单一系统均无法处理的综合数据清洗及其他处理工作。这里，输入文件来自多个数据源，通过预处理，将预处理后的结果放入 Hadoop 文件系统，使得 Hadoop 平台对应地作为数据源服务器，为下一步的数据清洗做准备。

（2）Hadoop 分布式计算

使用 Hadoop 分布式环境来实现集群的存储及计算，通过 HDFS 分布式文件系统实现对数据文件的存储和管理，通过 Map/Reduce 运行机制实现并行化。这里一方面实现对清洗数据的清洗，同时也负责中间输出文件的保存及管理，另一方面为下面的数据清洗引擎模块提供了分布式计算的运行机制，需要在 Hadoop 环境下实现算法任务的并行化处理。

（3）数据清洗引擎

这里是整个方案核心功能实现的模块。其主要通过基于 Hadoop 的分布式孤立点挖掘算法，对整个数据集进行清洗挖掘，找出不合理的属性值，并执行相应的数据清洗动作，最终将清洗后的数据通过接口或其他方式输出。这些处理结果可以传递给任意现有的与 Hadoop 无关的企业系统做进一步处理。这里，主要包括几个子功能模块，即数据加载模块、分布式孤立点挖掘算法模块、结果存储模块。

- 数据加载模块

由于数据是根据属性进行清洗的。在第一步中，我们已经将所有数据的各种属性均导入进来。这些属性分析有一些是需要分步进行的，故这里需要对数据进行再加载工作，将指定属性的数据加载到指定的 Hadoop 目录文件，输入数据来自于 HDFS 文件，输出数据也存放于 HDFS 文件。

- 分布式孤立点挖掘算法模块

由于我们主要是针对海量数据清洗的属性清洗的，这里采用孤立点挖掘算法，通过 Hadoop 分布式环境，找出异样的属性值，并结合相应的清洗规则继续分析处理。

- 结果存储模块

这里，将处理后的中间数据或最终数据结果，都通过该模块指定存放到对应的 HDFS 文件系统中，同时提供接口或其他方式，将清洗后的数据用于更高层的数据处理。

根据以上的方案，对应有以下的方案流程，如图 3.5 所示。

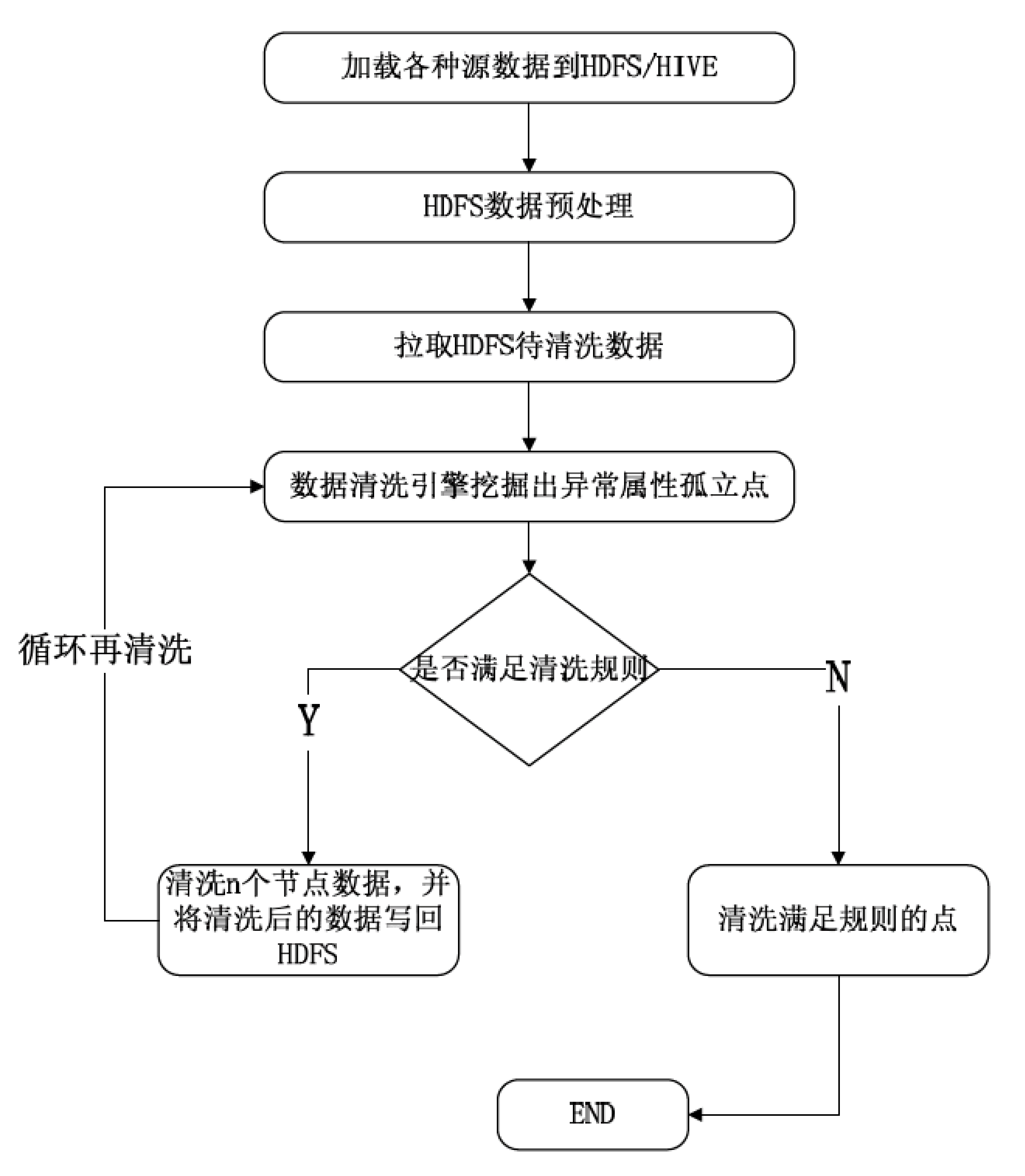


图 3.5　Hadoop 数据清洗流程图

分布式数据清洗方案的流程图主要包括以下步骤：

（1）首先通过 Sqoop 将各种异构数据源的数据加载到 Hadoop 分布式文件系统中。

（2）将 HDFS 的待清洗数据、参数 N 和 K 作为数据清洗引擎算法的输入。数据清洗引擎根据输入，求出本轮挖掘出的 N 个孤立点。

（3）对于挖掘出的 N 个孤立点则为清洗的候选点。我们需要判断这些孤立点是否满足清洗规则。若所有的 N 个点都满足清洗规则，则将 N 个数据点都根据清洗规则进行清洗，然后将清洗后的数据写回 HDFS，重新将新的 HDFS 数据、N 和 K 作为新的一轮数据清洗引擎的输入。

（4）若 N 个点只有若干个点满足清洗规则，由于孤立点的输出是根据距离优先原则，孤立性越明显，就排在越前面。故若只有若干个点满足清洗规则，则表示 N 个点中有 r 个点满足清洗规则，后面的 N−r 个点则已经不满足清洗规则，此时表明已经不需要进行下一轮新的数据清洗了，因为下一轮结果出来后，前面的几个点也是这 N-r 个点。故只需要简单地清洗 r 个点，写回 HDFS，便可以结束本次数据清洗方案的过程。

（5）若 N 个点中没有点满足清洗规则，与步骤（4）一样的分析，不需要进入新的一轮迭代数据清洗，同时也不需要进行数据清洗操作，直接借宿本次数据清洗方案的过程。

3.5 ETL 现状与发展

3.5.1 数据 ETL 简介

数据 ETL（Data Extraction, Transformation and Loading）是用来实现异构多数据源的数据集成的一个工具，它是数据仓库、数据挖掘以及商业智能等技术的基石。

ETL 软件（工具）的功能包括：

- 数据的抽取。从不同的网络、不同的操作平台、不同的数据库及数据格式、不同的应用中抽取数据；
- 数据的转换。数据转化（数据的合并、汇总、过滤、转换等）、数据的重新格式化和计算、关键数据的重新构建和数据总结、数据定位；
- 数据的加载。跨网络、操作平台，将数据加载到目标数据库中。

数据 ETL 是构建数据仓库的第一步，难点在于多源数据清洗、沉淀。对海量数据而言，人工处理不现实，故自动化数据清洗受到工业界的广泛关注。为了保证数据质量，需要定义和判断错误类型；查找并标示错误实例；修改没有发现的错误。由于这些问题比较凌乱而显得难以采用通用的方法进行处理，大多数研究工作都针对特定领域的数据集，或者是对不同性质的异常数据进行通用处理。

目前国内外关于数据清洗领域的研究非常活跃，主要涉及以下几个方面：

- 研究高效的数据异常检测算法以避免扫描整个庞大的数据集；
- 在自动化异常检测和清洗处理中增加人工判断处理以提高处理精度；
- 数据清洗时对海量数据集进行并行处理；
- 如何消除合并后数据集中的重复数据；

- 建立一个通用的领域无关的数据清洗框架;
- 研究模式集成问题。

已有研究为数据 ETL 积累了丰富的脏数据处理经验，提出了诸多数据清洗算法：脏数据预处理、排序邻居方法、优先排队算法、多次遍历数据清理方法、增量数据清理、采用领域知识进行清理、采用数据库管理系统的集成数据清理算法等。这些算法大多可运用到数据 ETL 的数据清洗过程中，极大地简化了数据 ETL 软件的实现，提升了最终软件的服务质量。

另一方面，数据仓库的发展则不断给数据 ETL 研究提供新课题。过去由于数据清洗与问题域的相关性很强，通用的数据清洗可能受到很大的限制，因此数据清洗方面的研究大都是针对具体应用、具体领域开展的，数据清洗框架的通用性很少有人关注。然而，数据仓库不断拓宽数据 ETL 应用领域，通用的清理方案必将受到越来越多的重视。在将多源数据导入数据仓库的过程中，数据 ETL 需要处理的是海量数据集，因此，增量式的数据抽取，清洗时增量式的数据异常检测、数据转换算法是必需的，而且对于算法的效率提出了愈来愈高的要求。

将数据从各种业务处理系统导入数据仓库是一个复杂的系统工程，数据 ETL 在此面临两个主要的挑战，其一为异构数据源的集成问题，其二为脏数据的检测与解决。虽然数据 ETL 作为数据仓库的预处理部分已经进入实用阶段，但这两个问题至今并没有得到很好的解决，成为业界研究的持续热点。

- 异构数据源集成问题。即数据集成，主要处理多数据源的异构问题。待集成数据源的异构性分为 4 个层次：系统、语法、结构和语义。系统级异构指不同的主机、操作系统和网络；语法级异构是指数据类型、格式的差异；结构级异构是指数据结构、接口和模式上的不同；语义级异构则是指在一定领域内专用的词汇意义的共享和交流。
- 脏数据的检测与解决。即“数据清洗”，用来有效地清除脏数据、保证数据质量。对于创建数据仓库及其后续工作，如数据挖掘等，需要保证数据的正确性、一致性、完整性和可靠性（Reliability），而目前的现存管理系统中的数据存在很多问题，容易造成脏数据，其原因有：滥用缩写词、惯用语、数据输入错误、数据中的内嵌控制信息、重复记录、丢失值、拼写变化、不同的计量单位和过时的编码等。

事实上，数据 ETL 需要解决的这两个问题并不存在十分清晰的划分边界，一般认为“数据集成”是“目的”，而“数据清洗”则是实现集成的主要手段，它们往往交织在一起，相互渗透。

3.5.2　基于 MapReduce 的 ETL 框架

最近几年，在处理 TB 和 PB 级数据方面，MapReduce 已经成为使用最为广泛的并行编程模型之一，本章设计的 ETL 框架便是基于 MapReduce 编程模型的。要在 MapReduce 模型下实现数据的 ETL 过程，需要对 MapReduce 的运行机制、连接算法和性能优化等诸多方面进行研究。目前，国内外 MapReduce 相关的研究成果主要有以下几方面。

（1）性能优化方面

2010 年，Shivnath Babu 研究了 MapReduce 的参数自动优化技术，该自动化技术不仅提高了 MapReduce 程序处理大数据集的性能，而且提高了缺乏 MapReduce 程序优化技巧的用户的生产效率。2010 年，Foto N. Afrati 等研究了在 MapReduce 中连接操作的优化技术，提出了链式连接和星型连接的优化算法。与 MapReduce 中传统连接的实现方式相比，该算法在处理大事实表与小维表的连接操作时性能尤佳。2011 年，Avrilia Floratou 把数据的列存储方式引入到 Hadoop 中，并实现了一个新的 HDFS 数据块分配策略以解决相关列的定位问题。

（2）连接算法方面

MapReduce 框架主要应用于大数据的分析，如日志数据的处理。处理日志时经常需要对数据进行过滤、聚集等操作，其中经常涉及数据的连接操作。但是，在 MapReduce 框架下实现数据的连接操作非常繁杂。目前的主要研究成果有日志处理的 Join 算法、Theta-Joins 算法和 Set-Similarity Joins 算法。

（3）编程模型改进方面

2007 年，Hung-chi Yang 等提出了 MapReduce-Merge 模型，该模型支持关系代数操作并实现了多个连接算法。2010 年，Tyson Condie 等提出 MapReduce Online 模型，该模型把 Online Aggregations 技术应用于 MapReduc 架构中，并以流水线方式处理数据，提高了数据处理的效率和响应速度。

1. ETL 数据处理流程

ETL 处理流程主要包括数据抽取、数据转换和数据加载。ETL 首先从不同的数据源中抽取出数据，数据被加载到数据仓库之前，要在 DSA 中对数据进行转换和清洗。

（1）数据抽取

数据抽取是指从异构数据源中获取符合要求的数据的过程。数据抽取过程会把源数据中不需要的字段过滤掉，并对源数据进行格式化和类型转换。数据抽取可以采用 PULL 和 PUSH 两种方式。PUSH 是指将源数据按照双方定义的数据格式抽取出来，再通过 FTP 或其他文件传送方式拷贝到 ETL 系统中。PULL 则是 ETL 程序直接访问数据源，获取数据的方式。在数据仓库创建过程中数据抽取模式主要分为全量抽取和增量抽取两种，下面对其分别进行介绍。

- 全量抽取

在这种模式下，加载数据仓库前，要将目标数据表完全清空，然后抽取程序抽取源数据中的所有记录。

- 增量抽取

增量抽取只抽取自上次抽取以来源数据中新增或发生变化的数据。在 ETL 过程中，由于增量抽取可以有效地降低后续阶段的数据量，因此应用更广。增量抽取的关键是通过一定的手段准确而快速地捕获到变化的数据，同时，又不能对现有的业务系统造成太大的压力。所以，与全量抽

取相比较，增量抽取方法更为复杂。

（2）数据转换

数据仓库的数据源形式多种多样，仅仅是数据库系统，就可以是 Sybase、Informix、Oracle、IBM 的 DB2 或者是 Microsoft SQL Server 等数据库系统中的一个或几个。在数据仓库中需要使用统一的格式存储数据，所以在创建数据仓库的过程中，需要将来源于不同平台的数据进行转换。数据转换可以在 ETL 过程中进行，也可以在数据抽取过程中利用关系数据库的特性进行。

（3）数据加载

数据加载是指从源业务系统中抽取的数据，经过转换后加载到目标数据仓库系统中的过程。不同的数据仓库提供商，都会有自己的数据加载工具以及深入编程的接口 API。对于用户而言，需要重点考察的是数据加载工具的加载性能，要求数据加载工具必须具有高效的加载性能。

2. ETL 框架的设计目标

基于 MapReduce 的 ETL 框架设计目标如下。

（1）灵活性

随着企业业务的不断复杂化和数据量的增长，ETL 流程也越来越复杂，使用可视化 ETL 工具中组件拖拽的方式无法灵活地设计 ETL 流程。手工编码型的 ETL 工具具有高度的灵活性，本章设计的 ETL 框架是基于编程方式的，用户可以调用相应的 API 或插入自己的 ETL 组件对其扩展来设计一个复杂的 ETL 流程。

（2）高性能

ETL 的性能必须能够确保在数据量非常大的情况下依然能够稳定地、快速地进行数据的处理，不会因数据量大而使 ETL 系统瘫痪。在不需要投入大量硬件成本的前提下，可以通过以下三点提高 ETL 处理的性能：依赖于 MapReduce 编程实现 ETL；对维度数据处理算法进行优化；优化 Hadoop 中数据块的分配策略。

（3）易扩展

首先，由于各个企业业务数据量的不同，所需要的集群规模也各异，ETL 工具应该具有较高的伸缩性以满足不同的需求。其次，越来越多的企业想实现自己的 ETL 系统，所以 ETL 框架应该方便用户进行二次开发，降低 ETL 系统的开发难度并减少 ETL 系统的开发周期。本章基于 MapReduce 设计的 ETL 框架可运行在超过 1000 个普通 PC 主机搭建的集群环境之上，并且，每增加一个节点，MapReduce 就能将差不多一台主机的计算能力加入到集群中，所以该框架具有较高的伸缩性。该 ETL 框架实现了通用的 ETL 功能，基于其编码实现 ETL 过程只需要考虑数据转换逻辑的具体实现，而不必关注周边功能，因此，该框架是易扩展的。

（4）易用性

ETL 工具应该方便没有分布式编程经验或者不了解 MapReduce 原理的 ETL 编程人员使用和对其扩展。该框架是基于 MapReduce 设计的，而 MapReduce 隐藏了并行计算、负载均衡等细节，

用户无须考虑分布式编程的逻辑设计与实现。ETL 框架封装了维度数据 ETL 和事实数据 ETL 的实现细节并对外提供 API，用户还可以插入自己设计的 ETL 组件，用户使用时只需调用相关的 API 即可使复杂的 ETL 流程转换为 MapReduce 任务。

3. ETL 框架设计

ETL 框架是基于 MapReduce 编程模型的，它主要由维度数据 ETL 模块、事实数据 ETL 模块和配置文件组成。为了优化 ETL 程序处理数据的性能，在该 ETL 框架中设计了源数据的切分策略。数据切分之后存储在 HDFS 的各个 DataNode 中，由 mapper 以一定的数据格式对源数据进行抽取。

配置文件是 ETL 流程的入口，并控制整个 ETL 的流程，维表和事实表是在配置文件中进行定义的。在配置文件中用户还可以自定义数据处理所需的转换函数和聚集函数，如果不需要这些操作，用户可不用配置 reducer，这样数据就不必划分给 reducer，减少了网络开销，提高了 ETL 处理的性能。

在该框架中，首先有一个 MapReduce Job 处理维度数据，然后另一 MapReduce Job 处理事实数据。事实数据的处理主要包括维度代理键查找、度量计算以及装载事实数据到数据仓库中。事实数据处理要在维度数据处理之后进行是因为处理事实数据时需要从维度数据中查找代理键。

在该 ETL 框架中，设计了两种维度数据的处理方式。一种是一个 map/reduce task 处理一个维度，比如，有三个维度和三个 map/reduce task，则每一个 map/reduce task 处理一个维度的数据集；另一种是，每个 map/reduce task 处理维度数据的一部分。下面基于 MapReduce 的 ETL 框架执行 ETL 过程的算法：

- 切分源数据集。
- 读取配置参数并初始化。
- 读取输入数据交给 map 函数处理。
- 处理维度数据并装载到数据仓库。
- 准备事实处理。
- 读取数据并在 mapper 中执行转换操作。
- 加载事实数据到数据仓库。

ETL 框架是由配置文件控制整个 ETL 的执行流程的，所有运行时的参数都存储在配置文件中，包括数据源的定义、源数据的切分策略、维表定义、事实表定义、大维表定义以及 mapper 和 reducer 的个数等。如果知道一个维度是大维度，那么用户以 bigdim 的形式配置该维度，这样 ETL 程序会选择对应的处理方式，可达到更好的性能和负载均衡。如果维度数据处理不需要聚集操作则可不用配置 reducer，那么 ETL 过程就省略了 Reduce 阶段，提高了 ETL 处理效率。

4. 数据切分策略

MapReduce 开始工作之前，Hadoop 需要将来自异构存储系统的大数据集进行切分，同时分发给各个 map/reduce task。每一个 task 按定义好的格式读取数据。针对数据集的特点设计一个好

的数据切分策略以降低网络传输的开销，提高 ETL 的效率。ETL 框架设有两种切分源数据的策略，它们是轮询切分策略和哈希切分策略。

（1）轮询切分策略

行号为 n 的数据分配给编号为（n mod nl_map）的 task，其中 nl_map 表示 task 的数量。这种策略把输入数据集平均地分配给各个 task，该策略适合由多个 task 同时处理维度数据的情况。

（2）哈希切分策略

该策略根据一个或多个属性的哈希值进行切分，属性哈希值相同的记录行被指派给同一个 task。如果有 nr_map 个 task，某一条记录的哈希值为 11，那么它将被分配给编号为（h mod nr_map）的 task。该策略适合同一个 task 处理相同属性值的所有记录行。

Hadoop 对源数据的切分和读取操作是由 InputFormat 类完成的。InputFormat 是一个抽象类，它包含 getSplits()和 createRecordReader()两个抽象方法。getSplits()确定对输入数据的切分原则，createRecordReader()则可以按一定格式读取相应数据。为实现对源数据的轮询切分和哈希切分策略，在 ETL 框架中设计了两个类，LineInputFormat 和 HasliInputFonnat。LineInputFormat 类是以轮询的方式切分数据，HashInputFormat 类是以哈希方式切分数据。它们都继承自 InputFormat 抽象类，并在 getSplits()方法中实现不同的切分策略。

5. 维度数据 ETL 设计

基于 MapReduce 的 ETL 程序对数据进行处理时，首先要处理维度数据，这样处理事实数据时才能通过 Lookup 转换操作获取其代理键。

ETL 框架使用 MapReduce 的 Map、Partition、Combine 和 Reduce 完成 ETL 流程。但是这些操作对用户是透明的，用户只需在配置文件中定义维表和事实表对象，并定制数据的转换操作即可完成维度数据和事实数据的装载。ETL 框架设有两种维度数据处理的方式：一种是 One Dimension One Task，简写为 ODOT，即一个维度的数据由一个 map/reduce task 处理；另一种是 One Dimension All Tasks，简写为 ODAT，即一个维度的数据由多个 map/reduce task 进行处理。

6. ODOT 处理方式

以 ODOT 方式处理维度数据的流程是：首先，ETL 程序按行读取源数据；然后，mapper 根据配置文件中维表的定义，对行数据做投影操作，去掉不需要的数据列，并格式化为 key/value 对，key 值相同的 key/value 对被分配到同一个 reducer 节点；最后，reducer 对数据做进一步处理。

在以 ODOT 方式对维度数据进行处理的 Reduce 阶段中，执行用户定制的转换函数。转换函数对数据进行转换操作之后，把数据插入数据仓库的维表中。插入一条维度记录时：

- 如果维表中不存在该条维度数据，则插入。
- 如果维表中存在该条维度数据且要插入的这条维度数据与其相比未发生变化，则什么都不做。
- 如果维表中存在该条维度数据且要插入的这条维度数据与其相比发生了变化，则分两种情况进行处理。第一种情况，如果该条数据是渐变维度类型的，则使用更新策略对其进行更

新；第二种情况，如果该条数据不是渐变维度类型的，则直接在历史数据上进行修改。

在一些情况下，维度数据可能会很大，比如渐变维度数据。因为更新渐变维度数据时需要添加新的记录行，如果更新频繁，则数据量会快速增长。如果使用 ODOT 方式处理维度数据，那么只有一个 map/reduce task 处理维度数据，这会影响整体 ETL 的处理性能。针对大维度数据的特殊情况，ETL 框架设计的 ODAT 维度数据处理方式，以提高维度数据处理的性能。维度数据以 ODAT 方式处理时，ETL 程序使用轮询方式划分 map 的输出结果，这样，各个 Reducer 将均等的获取数据，各个 reducer 处理每个维度的一部分数据，这样充分发挥了分布式处理的优势。

由于各个 task 并行地处理一个维度数据，那么最后的维度数据可能出现代理键重复或渐变维度数据不正确等问题。为解决这些数据问题，这里设计了两种处理方式，使用全局代理键时的 Post-fixing 和使用私有代理键时的 Post-fixing。其中，Post-fixing 是指当所有的 task 处理完维度数据并装载到数据仓库后，对问题数据进行统一的修改过程。

- 使用全局代理键时的 Post-fixing。即由 Hadoop 平台统一管理和分配维度数据所需的代理键，这样不会发生代理键重复的情况，但 Lookup 属性值可能会重复。当所有的 task 处理完维度数据后，再进行 Post-fixing 处理。
- 使用私有代理键时的 Post-fixing。每个 task 节点各自管理和分配维度数据所需的代理键，最后维度数据的代理键会有重复的现象，而且 Lookup 属性也可能会重复。当所有的 task 处理完维度数据后，再进行 Post-fixing 处理。

7. Post-fixing 模块设计

为解决 ETL 过程中出现的数据错误问题，可以设计 Post-fixing 模块。当维度数据加载到数据仓库之后，对其进行 Post-fixing 过程以纠正问题数据。对问题数据的纠正主要分为 4 步：

- 为有重复代理键的数据行分配新的代理键。
- 更新关联维表的外键。
- 删除业务键属性和外键属性值重复的数据行。
- 如果是 SCD 类型的维度，则修改 SCD 属性值。

Post-fxing 对问题数据进行纠正时需要分两种情况解决问题，即各个 mapReduce task 使用的是全局代理键策略还是私有代理键策略。

8. 事实数据 ETL 设计

ETL 框架中，第二阶段是对事实数据的处理，主要包括查找维度代理键、对度量做聚集操作并把处理后的数据装载到数据仓库中。查找代理键时，为 ETL 框架设计了 Lookup 转换策略。以 Lookup 转换方式查找维度数据代理键的流程是：首先，Lookup 函数把整个维度数据加载到主存中；然后，对维表中的业务键与源数据表中相对应的业务键进行映射；最后，Lookup 返回维度数据的代理键。与维度数据 ETL 类似，事实表同样是在配置文件中进行定义的。处理事实数据时，首先从配置文件中获取事实表的定义，根据事实表的定义，获取事实表所引用的维度，分别使用

Lookup 转换获取维度数据的代理键。然后根据用户配置的转换函数，对数据进行转换操作。最后把数据插入到维表中。

读数据，获得配置文件的过程可作为 Map 函数或 Reduce 函数。如果没有聚集操作，可把该函数配置为 Map 函数，并省略 Reduce 过程，则可取得更好的性能。如果需要聚集操作，则把该函数配置为 reduce 函数。这种设计方式不但灵活而且性能好。

9. 维度 ETL 优化

进行增量加载时，如果把大量维度数据从 mapper 划分给 reducer，效率会很低。假设加载之前根据业务键以哈希方式对数据进行分割，产生三个维度数据文件 D1、D2 和 D3，每个文件中的维度数据具有相同的业务键，这三个文件分别在数据节点 Nodel、Node2 和 Node3 上。使用相同的方式对增量数据进行划分，假设数据被分割为两个文件，S1 和 S3。D1 和 S1 的业务键值相同，D3 和 S3 的业务键值相同。当在 HDFS 中创建 S1 和 S3 时，如果根据块分配策略能使 Sl 和 D1 放在同一个数据节点上，S3 和 D3 放在同一个数据节点上，那么只通过 Map 阶段在本地即可完成维度数据的增量加载，而不需再把维度数据划分给 reducer，大大减少了网络开销。

从 Hadoop 0.21.0 开始，用户可以通过配置属性 dfs.block.replicater .classname 定制块的分配策略，这为实现块分配策略算法的优化提供了可能。这里设计的优化算法是根据扩展名定位文件，即文件扩展名跟正则表达式匹配的文件放在一起，正则表达式可在 name node 配置文件 core-site.xml 中进行定义。对 Hadoop 中块分配策略的优化算法核心思想如下。

当 NameNode 节点启动时，首先创建哈希目录 M，它存放数据节点和与之关联的节点中块信息的映射，比如，文件扩展名与正则表达式相匹配的文件块的总数目。当向 HDFS 中写入一个数据块时，首先要向 NameNode 询问，为数据块选择目标节点以及块的副本。NameNode 检查它的文件名是否与正则表达式相匹配，如果其文件名与正则表达式不匹配，NameNode 使用 HDFS 默认的策略为其选择目标节点。如果该数据块的文件名与正则表达式匹配，NameNode 则根据 M 中的统计信息选择为其目标节点。若 M 是空，则表明写入的是某个节点的第一个数据块，那么 NameNode 根据 HDFS 默认的块分配策略为其选择目标节点并更新 M；若 M 不是空，分配策略是，选择那些块的数量最多并且有足够空间的节点（该算法根据块的数量值对 M 排序，并把各个节点放进一个队列 Q），当 Q 中没有满足要求的节点个数时，NameNode 随机选择一个空闲节点，被选择的每个节点需要检查它是否满足选择标准以及是否有充足的空间。

对基于 MapReduce 的 ETL 框架的易用性和性能进行实验测试，首先，Hadoop 基于廉价的硬件即可搭建一个高性能的计算机群；然后，模拟数据集的规模从 40GB 到 320GB 不等；最后，与 Hive 中的 ETL 工具进行了对比分析，主要从维度数据装载、数据装载、大维度数据装载和事实数据装载 4 个方面对该 ETL 工具和 Hive 提供的 ETL 工具的处理时间进行了对比分析。实验过程中发现，使用 Hive 实现 ETL 程序需要复杂的脚本编程，而该 ETL 框架的使用更为简单，大大减少了代码编写量。而且实验结果显示，基于该 ETL 框架实现的 ETL 工具的处理性能在时间上要比 Hive 节约 70%左右。

随着对大数据价值认识的深入，大数据的典型特征又加入了一个价值维度，用以描述大数据的价值。在现实应用中，数据量大的数据并不一定有很大的价值。大数据带来的潜在经济价值和

社会价值巨大，但这些价值必须通过数据的有效整合、分析和挖掘才能释放出来。数据的整合是建立数据仓库的必要工作，针对结构化数据的整合有很多解决方案和软件工具。目前的挑战是非结构化数据的融合和整合，如文本数据、图像数据、信号数据、音频数据、视频数据等。

大数据带来的变化之一是对象的属性越来越多，虽然表达对象的信息越来越丰富，但成千上万的属性也造成巨大的维度灾难。与此同时，这种超高维数据也带来其他一些问题，如复杂数据类型问题、噪声和默认值问题、分布不平衡问题、属性相关问题等。这些问题虽然在一般性数据分析中普遍存在，但超高维数据使得这些问题更难处理。网络社会化文本数据，如微博数据，就属于这类大数据，表达微博内容的关键词属性可以有几万个，而处理的微博数量也是百万或千万级。

超高维数据不适合用传统的全空间方法来分析，因为超高维数据带有很大的稀疏性，对象簇和类别的表达体现在部分属性子集，较有效的分析方法是采用子空间方法。同时，由于数据的复杂性，单一的数据挖掘模型，如决策树模型，难以满足应用的精度要求，必须采用多个模型的集成学习方法建立聚类或分类的集成模型，通过多个单一模型的综合结果做出最后的决策。

大数据分析的另一科学问题是超过千万或亿的数量级后，这样大的输入数据远远超出大多数服务器的内存，更不用说在单一服务器上用复杂的迭代或递归数据挖掘算法进行建模和挖掘。因此，现有的数据挖掘软件和大多数传统的分类和聚类等算法无法处理这个规模的数据。

大数据分析与挖掘的另一科学问题是分析方法和分析手段落后。当前普遍采用的数据挖掘建模方法是样本→建模→测试三步骤方法，建模的过程由算法自动完成，模型建好后，用户对模型进行测试，结果不满意，改变训练数据和算法参数，由算法自动产生新的模型。这种方法不适用于大数据分析，因为数据大，算法建模的时间较长，多次重复建模步骤使计算成本和能耗加大。因此，必须研究新的大数据分析方法。

提高大数据分析与挖掘的效率和效果的方法之一是改变建模的全自动过程，实现大数据建模人机交互，让分析人员的领域知识融入到建模过程中，通过人机交互获得优化模型。实现大数据建模过程人机交互需要解决两大关键技术是交互式数据挖掘算法和数据及模型可视化。交互式算法在建模过程中生成大量中间结果，用可视化技术展现给分析人员，分析人员可以通过观察分析建模的阶段性结果，调整算法参数或输入数据，指引交互式算法向优化模型的方向计算。

“大数据”时代充满了机遇与挑战，谁能够最快地习惯这种新形式下的数据模式，熟悉掌握处理这种数据处理方法，谁就会在之后的信息战中占得先机，取得主动权。

3.5.3 ETL 工具

1. ETL 工具简述

ETL 所完成的工作主要包括三方面。首先，在数据仓库和业务系统之间搭建起一座桥梁，确保新的业务数据源源不断地进入数据仓库；其次，用户的分析和应用也能反映出最新的业务动态，虽然 ETL 在数据仓库架构的三部分中技术含量并不算高，但其涉及大量的业务逻辑和异构环境，因此在一般的数据仓库项目中 ETL 部分往往也是牵扯精力最多的；第三，如果从整体角度来看，

ETL 主要作用在于屏蔽了复杂的业务逻辑，从而为各种基于数据仓库的分析和应用提供了统一的数据接口，这也是构建数据仓库最重要的意义所在。正确选择 ETL 工具，可以从 ETL 对平台的支持、对数据源的支持、数据转换功能、管理和调度功能、集成和开放性、对元数据管理等功能出发。

随着各种应用系统数据量的飞速增长和对业务可靠性等要求的不断提高，人们对数据抽取工具的要求往往是将几十、上百个 GB 的数据在有限的几个小时内完成抽取转换和装载工作，这种挑战势必要求抽取工具对高性能的硬件和主机提供更多支持。因此，可以从数据抽取工具支持的平台，来判断它能否胜任企业的环境，目前主流的平台包括 SUN Solaris、HP-UX、IBM AIX、AS/400、OS/390、Sco UNIX、Linux、Windows 等。

对数据源支持的重要性不言而喻，因此这个指标必须仔细地考量。首先，我们需要对项目中可能会遇到的各种数据源有一个清晰的认识；其次对各种工具提供的数据源接口类型也要有深入了解，比如，针对同一种数据库，使用通用的接口（如 ODBC/JDBC）还是原厂商自己的专用接口，数据抽取效率都会有很大差别，这直接影响到能不能在有限的时间内完成 ETL 任务。

数据转换是 ETL 中最令人头疼的问题，由于业务系统的开发一般有一个较长的时间跨度，这就造成一种数据在业务系统中可能会有多种完全不同的存储格式，甚至还有许多数据仓库分析中所要求的数据在业务系统中并不直接存在，而是需要根据某些公式对各部分数据进行计算才能得到。因此，这就要求 ETL 工具必须对所抽取的数据进行灵活的计算、合并、拆分等转换操作。通常情况下，我们遇到的 ETL 转换要求包括：字段映射；映射的自动匹配；字段的拆分；多字段的混合运算；跨异构数据库的关联；自定义函数；多数据类型支持；复杂条件过滤；支持脏读；数据的批量装载；时间类型的转换；对各种码表的支持；环境变量是否可以动态修改；去重复记录；抽取断点；记录间合并或计算；记录拆分；抽取的字段是否可以动态修改；行、列变换；排序；统计；度量衡等常用的转换函数；代理主键的生成；调试功能；抽取远程数据；增量抽取的处理方式；制造样品数据；在转换过程中是否支持数据比较的功能；数据预览；性能监控；数据清洗及标准化；按行、按列的分组聚合等。

由于对数据抽取的要求越来越高以及专业 ETL 工具的不断涌现，ETL 过程早已不再是一个简单的小程序就可完成的，目前主流的工具都采用像多线程、分布式、负载均衡、集中管理等高性能、高可靠性与易管理和扩展的多层体系架构。因此，这就要求 ETL 在管理和调度功能上都具备相应的功能。管理和调度的基本功能包括：抽取过程的备份与恢复；升级；版本管理；开发和发布；支持统一以及自定义的管理平台；支持时间触发方式；支持事件触发方式；支持命令行执行方式；支持用户对计算机资源的管理和分配；负载均衡；文档的自动生成；调度过程中能否执行其他任务等。

随着数据仓库技术在国内应用的不断深入，许多开发商希望不向用户提供 ETL 工具的原来操作界面，而是将其一些主要功能模块嵌入到自己的系统或其他厂商的系统中，因为在大多数情况下一般项目只会用到 ETL 工具的少数几个功能，同时也没有必要给用户提供那么复杂的操作环境，其结果反而使用户容易产生操作错误。上述问题就要求 ETL 工具提供很好的集成性和开放性，可以从几方面考量：与 OLAP 集成；与前端工具集成；与建模工具集成；开放的 API 可将产品集成到统一界面；是否能调用各种外部应用，包括存储过程、各种流行语言开发的应用程序等；

是否支持客户化定制的转换过程；是否支持与统计分析工具的集成等。

元数据是关于数据的数据，对于 ETL 来说尤其重要。ETL 中大量的数据源定义、映射规则、转换规则、装载策略等都属于元数据范畴，如何妥善地存储这些信息已经关系到 ETL 过程能否顺利完成而且影响到后期的使用和维护。任何业务逻辑的微小改变最终都落实为相应元数据的调整，初期没有一个完善的元数据管理功能而后期做类似调整几乎是不可完成的任务。基于元数据的重要性，国际组织提出一些统一的元数据存储标准，比较知名的如 CWM 等，为不同厂商工具之间互操作提供了可能性，相信也是今后的发展趋势。

针对 ETL 的元数据管理包括：元数据存储的开放性；元数据存储的可移植性；提供多种方式访问元数据；元数据的版本控制；支持开放的元数据标准；支持 XML 进行元数据交换；支持分布式的元数据访问和管理；生成元数据报表；对于 ETL 过程的冲突分析；基于元数据的查询功能；元数据的广播和重用；对于 ETL 过程的流程分析等。

目前市场上主流的 ETL 工具可以分为两大类：一类是专业 ETL 厂商的产品，这类产品一般都具备较完善的体系结构和久经考验的产品，产品功能之复杂和详尽，往往能令初次接触的人瞠目，但其高昂的价格也会使一般用户望而却步；另一类是整体数据仓库方案供应商，他们在提供数据仓库存储、设计、展现工具的同时也提供相应的 ETL 工具，这类产品一般对自己厂商的相关产品有很好的支持并能发挥出其最大效率，但结构相对封闭，对其他厂商产品的支持也很有限。

专业 ETL 厂商和产品有：OWB（Oracle Warehouse Builder）、ODI（Oracle Data Integrator）、Informatic PowerCenter（Informatica 公司）、AICloudETL、DataStage（Ascential 公司）、Repository Explorer、Beeload、Kettle、DataSpider、ETL Automation（NCR Teradata 公司）、Data Integrator（Business Objects 公司）、DecisionStream（Cognos 公司）。

2. 主流 ETL 工具

做 ETL 产品的造型，需要从成本、人员经验、案例和技术支持来考量。在此介绍几种主流 ETL 产品，包括 Ascential 公司的 DataStage、Oracle 的 Oracle Warehouse Builder（OWB）、Informatica 公司的 PowerCenter 和 NCR Teradata 公司的 ETL Automation。

（1）DataStage

DataStage 是由 IBM 公司开发的，是一套专门对多种操作数据源的数据抽取、转换和维护过程进行简化和自动化，并将其输入数据集市或数据仓库目标数据库的集成工具。DataStage 使用图形化概念来构建数据集成并拥有多个版本，如服务器版本和企业版本。DataStage 支持工业标准和连接需求，可以解决商业上的实时问题。DataStage 的体系架构是开放和可扩展的，这意味着新技术可以迅速被用于具体的工具需求，从而加快实施，减少风险并增加操作效率。DataStage 能够处理多种数据源的数据，包括主机系统的大型数据库、开放系统上的关系数据库和普通的文件系统等，它所能处理的主要数据源包括：

- 大型主机系统数据库。IMS，DB2，ADABAS，VSAM 等；
- 开放系统的关系数据库。Informix，Oracle，Sybase，DB2，Microsoft SQL Server 等；
- ERP 系统。SAP/R3，PeopleSoft 系统等；

- 普通文件和复杂文件系统。FTP 文件系统，XML 等；
- Web 服务器系统。IIS，Netscape，Apache 等；
- Email 系统。Outlook 等；

DataStage 可以从多个不同的业务系统中，从多个平台的数据源中抽取数据，完成转换和清洗，装载到各种系统里面。其中每步都可以在图形化工具里完成，同样可以灵活地被外部系统调度，提供专门的设计工具来设计转换规则和清洗规则等，实现了增量抽取、任务调度等多种复杂而实用的功能。其中简单的数据转换可以通过在界面上拖拉操作和调用一些 DataStage 预定义转换函数来实现，复杂转换可以通过编写脚本或结合其他语言的扩展来实现。DataStage 提供调试环境，可以极大提高开发和调试抽取、转换程序的效率。

DataStage 是基于客户机/服务器的数据集成架构，优化数据收集、转换和巩固的过程。它提供了一套图形化的客户工具，包括：

- Designer（设计器）：创建执行数据集成任务 Job 的同时，对数据流和转换过程创建一个可视化的演示。
- Manager（管理器）：对每个工程的各个单元，包括库表定义、集中的数据转换、元数据连接等对象进行分类和组织。
- Director（控制器）：为启动、停止、监视作业提供交互式控制。
- Administrator（管理器）：在服务器端管理 DataStage 的项目和使用者权限的分配，在 v8.1 版本中，Manager（管理器）已经取消，合并到 Designer。

DataStage 具有易操作性，大容量、复杂事物的处理几乎是自动实现的，不需要添加额外编码，而且可以快速地返回结果。可以快速地、无缝地与大部分的主流应用程序、数据库和信息系统集成；可以灵活响应，快速地进行商业策略需求的演变；可以对基于项目的企业系统和应用程序进行补充或满足全集成方案的需求。

作为最强大的 ETL 工具，DataStage 提供了独一无二的功能：支持大容量数据的收集、集成和转换，数据结构可以是简单的，也可以是非常复杂的；实时处理数据，可以是每日、每周或是每月到达的数据。DataStag 允许企业通过高性能的处理海量数据集的方式来解决大范围的商业问题，通过扩展多处理器硬件平台的并行处理能力，DataStage 企业版可以满足不断增长的数据和实时的并行处理的要求。DataStage 允许在一个简单的作业（job）中，支持虚拟的数量限制的异质数据源和目标数据源 包括文本文件 XML 中的复杂数据结构 ERP 系统（如 SAP 和 PeopleSoft）、绝大多数的数据库（包括并行数据库）、Web 服务和 SAS。DataStage 的主要特点如下：

- 数据源连接能力。数据整合工具的数据源连接能力是非常重要的，这将直接决定它能够应用的范围。DataStage 能够直接连接非常多的数据源，包括：文本文件、XML 文件、企业应用程序、几乎所有的数据库系统、Web Services、SAS、WebSphere MQ。
- 多国语言支持（NLS）。DataStage 能够支持几乎所有编码，以及多种扩展编码（IBM、NEC、富士通、日立等），可以添加编码的支持，DataStage 内部为 UTF8 编码。
- 并行运行能力。ETL Job 的控件大多数都支持并行运行，此外 DataStage 企业版还可以在

多台装有 DataStage Server 的机器上并行执行。这也是传统的手工编码方式难以做到的。这样，DataStage 就可以充分利用硬件资源。而且，当硬件资源升级的时候也不用修改已经开发好的 ETL Job，只需要修改一个描述硬件资源的文件即可。并行执行能力使 DataStage 所能处理数据的速度可以得到趋近于线性的扩展，轻松处理大量数据。

- 便捷的开发环境。DataStage 的开发环境是基于 C/S 模式的，通过 DataStage Client 连接到 DataStage Server 上进行开发。这里有一点需要注意，DataStage Client 只能安装在 Windows 平台上（在 Win2000/XP 上运行过）。而 DataStage Server 则支持多种平台，比如 Windows、Solaris、Redhat Linux、AIX、HP-UNIX（在 WinXP/Solaris8 上运行过）。DataStage Client 有 4 种客户端工具，分别是 DataStage Administrator、DataStage Designer、DataStage Manager、DataStage Director。下面介绍这几种客户端工具在 DataStage 架构中所处的位置以及它们是如何协同工作来开发 ETL Job 的。
- 命令行形式的运行。ETL Job 支持在 DataStage Server 端用命令行形式调用，可以用 dsadmin 命令来管理 DataStage 的 Project，包括 Project 的新建、删除以及一些环境变量的增删（DataStage 7.5.1 下未能通过 dsadmin 来设置全局 NLS 和一些项目属性）。使用 dsjob 命令，能够同步或非同步地运行 DataStage 的 Job，并传递需要的 Job 参数，能够检查 Job 运行的状态，并能恢复 Job 的运行状态。
- DataStage 的不足。以上都是说 DataStage 优点，但实际上 DataStage 也有不少缺点和不足，这些不足点会直接影响到能否采用 DataStage 来达到客户或设计要求。如一些高级控件的功能不够全面，在实际应用时，会出现不能完全利用 DataStage 提供的控件来满足要求，如 Sybase 的 BCP、DataStage 的 Sybase BCP 控件只支持导出、无法支持导入。

（2）Oracle Warehouse Builder

Oracle Warehouse Builder（OWB）是 Oracle 公司于 1998 年推出的一个用于帮助企业构建数据仓库的集成工具。OWB 将从前各自分离的产品提供的功能集成到一个公共的环境，它提供了一个易于使用的图形环境，用于快速设计、部署和管理商务智能系统。其功能包括：数据模型构造和设计、数据提取、移动和装载（(ETL)、元数据管理、分析工具的整合以及数据仓库的管理。OWB 提供了一个框架将数据仓库的各个部分包括关系数据库服务器、多维数据库服务器和前端分析工具相结合，从而产生了一个紧密集成、全面的数据仓库和商业智能（BI）解决方案。OWB 减少了企业建设数据仓库的时间、成本和工作量。开发项目小组成员现在可以在一个单一的环境来实施和管理复杂的数据仓库系统。

OWB 的核心领域为企业元数据管理、企业数据集成、完整的系统设计、集成的质量以及开放性。OWB 具有如下特点：

- 全面的数据仓库功能。与其他工具不同，OWB 更适合具有智能的数据仓库的特殊要求，从它的设计和生成功能到从多个来源提取数据和向目标数据仓库装载数据，OWB 的每一个方面都降低了企业数据仓库项目的复杂性。
- 强调数据仓库的管理。目前市场上的很多数据仓库工具产品往往只强调数据仓库的生成过程，而忽略了对数据仓库进行管理的需要。OWB 在提供强大的数据仓库生成功能的同时，

更强调对企业数据仓库的管理。需要说明的是，OWB 的管理功能可以被集成到 Oracle Enterprise Manager（OEM）中，从而提供一个无缝的企业管理工具。

- 支持复杂的提取、转换和传输过程。OWB 通过 Oracle 提供的透明网关技术，支持从 Oracle 数据库、ODBC 数据源和大型主机系统中快速提取和有效装载数据。OWB 还支持多种类型的数据转换方法，并能对转换过程进行记录，从而不断强化定义的商业规则，保证被转换和装载数据的完整性。
- 利用数据库服务器提高性能。Oracle 一直不断地改进和提高其业界领先的数据库产品。Oracle 在与数据仓库相关的主要方面提供了新的功能。OWB 充分利用了 Oracle 的新功能，包括汇总管理、数据分区和索引能力。
- 与前端分析工具紧密集成。数据仓库的真正作用在于信息的分析，数据仓库的构建工具只有同前端分析工具集成，才能称为完整的平台。OWB 支持专用于分析的多维模型，可以生成符合标准的元数据（MetaData），可以和复杂的分析工具如 Oracle Express、Discoverer 相集成。
- 开放、可延伸的框架。OWB 提供软件开发包（SDK）供客户和合作伙伴使用，通过使用 SDK 可以很方便地扩展 OWB 的功能，定制客户化的数据转换程序。SDK 包括对外公开的 API 和一个开放的数据模型，第三方厂商可以很方便地将自己的应用与 OWB 相结合。

Warehouse Builder 主要由以下几部分组成：

- OWB User Interface。一个图形化的、采用面向对象技术，基于 Java 的框架，实现从任何平台管理数据模型建立和数据仓库环境的快速构造。
- OWB Repositorya OWB Repository 包含遵循 Common Warehouse Meta data 标准的元数据。OWB 元数据用于建立数据仓库，提供和 Oracle 数据库服务器、Express Server 以及 Discoverer 的集成，包括设计时和运行时两部分。
- OWB Runtime Platform Serviceo OWB 的核心，整个系统的运行时环境，完成已配置完成的 ETL 过程。
- OWB Software Development Kit。使用 SDK，用户和合作伙伴可通过集成他们自己的数据抽取程序实现 OWB 功能的扩展。

（3）Informatica PowerCenter

PowerCenter 是 Informatica 公司开发的，提供专注于复杂的数据集成项目必需的工具和数据服务平台，用来访问、集成和传递数据。使用 Informatica PowerCenter 企业能够通过一次建立、任意部署的方法从事多个不同的集成项目，允许更多的时间和资源花费在企业的业务上，而不是企业的业务整合上。

Informatica PowerCenter 主要由 2 个 Server、5 个 Client 组成。

- 2 个 Server

Informatica Repoitory Server：资料库 Server，管理 ETL 过程中产生的元数据。

Informatica Server Engine：ETL 引擎，负责抽取、转换和装载数据。

- 5 个 Client

PowerCenter Designer：设计开发环境，定义源及目标数据结构，设计转换规则，生成 ETL 映射。

Workflow Manager：合理的实现复杂的 ETL 工作流，基于时间、事件的作业调度。

Workflow Monitor：监控 Workflow 和 Session 运行情况，生成日志和报告。

Repository Manager：资料库管理，包括安全性管理等。

Repository Server Administrator Console：资料库的建立与维护。

Informatica PowerCenter 提供对应用和数据源的支持，包括对 ERP 系统的支持，对 CRM 系统的支持等，同时 PowerCenter 具有极好的可扩展性和可伸缩性以及强大的数据并行处理能力，这些特点使其可以较好地满足企业数据集成及应用集成的需要。

（4）ETL Automation

继续要说的是 ETL Automation，它和前面几种产品的体系架构有所差异。与其说它是 ETL 工具，不如说是提供了一套 ETL 框架。它没有将注意力放在如何处理“转换”这个环节上，而是利用 Teradata 数据库本身的并行处理能力，用 SQL 语句来做数据转换的工作，其重点是提供对 ETL 流程的支持，包括前后依赖、执行和监控等。

这样的设计和 Datastage、Powercenter 风格迥异，后两者给人的印象是具有灵活的图形化界面，开发者可以傻瓜式地处理 ETL 工作，它们一般都拥有非常多的“转换”组件，例如聚集汇总、缓慢变化维的转换。而对于 Teradata 的 ETL Automation，有人说它其实应该叫做 ELT，即装载是在转换之前的。的确，如果依赖数据库的能力去处理转换，恐怕只能是 ELT，因为转换只能在数据库内部进行。从这个角度看，Automation 对数据库的依赖不小，似乎是一种不灵活的设计。也正是这个原因，考虑它的成本就不单单是 ETL 产品的成本了。

其实，在购买现成的工具之外，还有自己从头开发 ETL 程序的。ETL 工作看起来并不复杂，特别是在数据量小、没有什么转换逻辑的时候，自己开发似乎非常节省成本。的确，主流的 ETL 工具价格不菲，动辄几十万；而从头开发无非就是费点人力而已，可以控制。至于性能，大多人是相信自己的，认为自己开发出来的东西知根知底，至少这些程序可以完全由自己控制。

ETL Automation 通过 ODBC 连接到 TeraData 数据仓库，在 TeraData 系统中建立一个数据库作为 ETL 记录库（Repository）。在 Repository 中记录 ETL Automation 中的作业定义、作业相关性以及关于作业执行的历史记录信息。ETL Automation 软件机制中提供了两个 GUI 接口的前端程序：一个是 ETLAdmin. jar，用来定义并管理在 ETL Automation 中的作业及作业关联性；另一个是 ETLMonitor. jar，用来监视作业的执行状态。

实际设计中，采用先数据转换、后数据抽取和数据加载的流程。首先从多个异构业务系统中将需抽取的数据转换成符合数据仓库要求的形式，同时将这些源数据抽取到一台或多台存放源数据的服务器中指定的文件夹下；然后 ETL Automation 调度抽取进程从源数据存放服务器中抽取数据到 ETL 服务器上，再调度加载进程将数据从 ETL 服务器上加载到数据仓库。

源数据存放的服务器采用 FTP 服务器，抽取进程采用 FTP 命令从服务器上下载数据到 ETL 服务器上。数据抽取有 PULL 和 PUSH 两种方式。PUSH 指由源系统按双方定义的数据格式，主

动将符合要求的数据抽取出来，形成接口数据表或数据视图供 ETL 系统使用；PULL 则是由 ETL 程序直接访问数据源来获取数据。这里采用 PUSH 方式，由源系统方将符合要求的数据抽取出放在 FTP 服务器中的指定文件夹下。

ETL 服务器是执行 ETL Automation 机制的服务器，该服务器上执行 ETL Automation 系统程序，用来控制整个 ETL 流程。

一个 ETL 流程除了在限定时间内完成业务系统源数据周期性的自动加载外，还要具有扩展能力。ETL Automation 工具可支持多部执行作业的服务器，设定整个 ETL Automation 中都使用到的 ET LAutomation Server，在该服务器中设定系统，用来将作业分类以方便管理。

在 FTP 服务器中每个数据文件对应一个标志文件，相当于一个握手信号，用来核对源数据生成时间和大小，以确保数据的正确抽取。首先执行 ETL 流程总调度系统下的总控制脚本，检查 etlslave-nt.p1 程序。该程序检查其对应目录下的所有文件，根据控带文件得到数据文件列表，更新任务开始时间，清除 ETL-JOB-STATUS 表中的任务，并在该表中插入新任务，执行作业的指令文件。这里，ETL-JOB-STATUS 表用来记录工作的执行状态。

执行数据抽取脚本时根据 ETL-FLG-DATA 表的顺序依次完成。在抽取脚本中，首先解析数据标志文件，连接、登录到 FTP 服务器；然后将传输模式改为二进制传输，并调用 Teradat 的 BTEQ 工具，清除数据仓库中的对应表，为采用刷新式加载做准备。

ETL Automation 根据目录下的任务名后缀来扫断是抽取脚本还是加载脚本，执行加载进程前，系统清除 ETL-RECORD-LOG 表记录，加载后向 ETL-RECORD-LOG 表插入记录。ETL- RECORD-LOC 表是数据加载的记录表。在数据加载脚本中，采用刷新式加载方式。首先调用 BTEQ 在数据仓库中建表；然后调用 Fastload 执行存储过程脚本，获取 ETL 服务器中指定目录下的数据文件，向已建空表加载数据。

ETL Automation 体系下有 3 个数据目录实现了数据抽取到加载的一致连贯性，分别是：receive 目录接收源系统所传送来的数据文件及控制文件；queue 目录存放准备要执行的作业所使用的数据文件及控带文件；process 目录存放正在执行中的作业所使用的数据文件及控制文件。

3.5.4 ETL 展望

数据 ETL 是数据仓库、数据挖掘以及商业智能等技术的基石，为企业决策与预测提供了基本素材，因而存在着广阔的发展空间。由于现实需求的强劲推动，数据 ETL 逐渐成为当前信息技术最为活跃的研究领域之一，呈现出通用化、高效化、智能化三大发展趋势。

数据是企业进行任何事务的前提，ETL 的目的正是提供综合且高品质的数据，因此它必然成为企业各类应用的基础，为众多的高层信息系统提供服务。具备良好的通用性是未来数据 ETL 软件占领市场的必要条件，这就要求它支持尽可能多的数据库管理系统（DBMS）、文件系统和数据采集、处理系统；能够跨网络、跨平台使用；具备良好的可扩展性，对于新的应用能够以较小的代价，通过预定的应用程序接口（API）或标准化语言接口编程实现互联。相关技术的发展，如元数据的标准化、程序逻辑与数据的统一化，为 ETI 提高通用性提供了动力。

数据 ETL 针对的是海量数据，效率极为重要，未来的 ETL 工具将是高效化的数据集成工具。

它必须具备高度的可伸缩性，不但能运行在昂贵的主机系统上，还能应用到工作站或 PC 机上。业界也将提供更加出色、高效的抽取、加载和清洗算法，增量的 ETL 算法将成为主流，真正避免重复集成；为了提高计算性价比，并行算法将领导潮流，集群计算、网格运算将为 ETL 提供廉价高效的计算资源。

未来的 ETL 将具备高度的智能，专家系统、机器学习、神经网络、人工智能（AI）技术等领域的成果将在此处得到广泛应用。数据源管理、ETL 规则定制、数据质量保证等工作都将由机器智能来完成。因此，当前手工或半手工的许多单调而繁重的数据集成任务将不复存在，ETL 工具的使用也会不断简化，普通用户能够运用智能工具轻松而高效地完成数据的集成与清洗工作。

数据 ETL 工具是数据仓库获取高质量数据的核心部分，根据决策需求将各种异构信息集中到数据仓库中，解决各种应用数据零散分布、品质低下的现状。数据 ETL 最大的挑战来自待集成多数据源的异构性，为此实施数据 ETL 过程通常被划分为模式集成与数据集成两个阶段，这样有利于将数据转化的逻辑规范和物理实现清晰分开方便管理，降低系统实现的难度。

3.6 本章小结

本章首先介绍了数据抽取方面的内容，侧重介绍了 Web 方向的数据抽取框架，简述了 Web 抽取面临的各种问题和解决方案。同时介绍了在大数据环境下，面对大量的非结构化数据的组织抽取，突出了非结构化数据的数据模型、属性获取策略等，提出了基于云计算的海量数据分析设计思路。第二部分侧重于数据清洗，首先简介数据抽取的背景意义，以及数据质量的标准、普遍的设计流程和框架。列举了通常遇到的问题和解决方案，给出在 Hadoop 环境下基于分布式的数据清洗设计。最后简述 ETL 的现状和发展，并给出常用的 ETL 工具介绍，以及未来发展前景展望。综合本章内容，可以为大数据实战奠定部分基础。

第 4 章 数据集成

在介绍数据获取、存储、抽取与清洗技术之后，相信各位读者已经对如何从海量数据中进行抓取、存储以及初步处理比较清楚了。海量数据意味着很高的价值，然而，只是对海量数据进行以上操作的话，离从海量数据中提炼出它们蕴含的价值还远远不够，本章将向大家讲述大数据提炼价值的一个关键步骤——数据集成。

在本章中，笔者将会对什么是数据集成技术，各大公司对数据集成技术的定义有什么区别，本书将怎么定义数据集成技术，数据集成技术有什么用处，数据集成技术的发展历程和分类，数据集成技术的研究现状以及各大公司是用什么工具如何实现数据集成技术的，在大数据背景下的数据集成技术又有哪些特点等问题逐一做出解答。本章将更多地对比各大公司在数据集成技术方面的认识和实现，并加以分析提炼出一个更为普遍的思路介绍给读者，在详细阅读本章后，希望读者能对数据集成技术有一个清晰明确的认识。

4.1 数据集成技术介绍

近几十年来，科学技术的迅猛发展和信息化的推进，使得人类社会所积累的数据量已经超过了过去 5 000 年的总和，数据的采集、存储、处理和传播的数量也与日俱增。企业实现数据共享，可以使更多的人更充分地使用已有数据资源，减少资料收集、数据采集等重复劳动和相应费用。但是，在实施数据共享的过程当中，由于不同用户提供的数据可能来自不同的途径，其数据内容、数据格式和数据质量千差万别，有时甚至会遇到数据格式不能转换或数据转换格式后丢失信息等棘手问题，严重阻碍了数据在各部门和各软件系统中的流动与共享。因此，如何对数据进行有效的集成管理已成为增强企业商业竞争力的必然选择。

由于现代企业的飞速发展和企业逐渐从一个孤立节点发展成为不断与网络交换信息和进行商务事务的实体，企业数据交换也从企业内部走向了企业之间；同时，数据的不确定性和频繁变动，以及这些集成系统在实现技术和物理数据上的紧耦合关系，导致一旦应用发生变化或物理数据变动，整个体系将不得不随之修改。因此，我们进行数据集成将面临着如何适应现代社会发展的复杂需求、有效扩展应用领域、分离实现技术和应用需求、充分描述各种数据源格式以及发布和进行数据交换等问题。

就大型企业和政府部门的信息化而言，信息系统建设通常具有阶段性和分布性的特点，这就导致“信息孤岛”现象的存在。“信息孤岛”造成系统中存在大量冗余数据、垃圾数据，无法保证数据的一致性，从而降低信息的利用效率和利用率。为解决这一问题，人们开始关注数据集成研究。数据集成的核心任务是要将互相关联的分布式异构数据源集成到一起，使用户能够以透明的方式访问这些数据源。集成是指维护数据源整体上的数据一致性、提高信息共享利用的效率；透明的方式是指用户无需关心如何实现对异构数据源数据的访问，只关心以何种方式访问何种数据。实现数据集成的系统称作数据集成系统（如图 4.1 所示），它为用户提供统一的数据源访问接口，执行用户对数据源的访问请求。

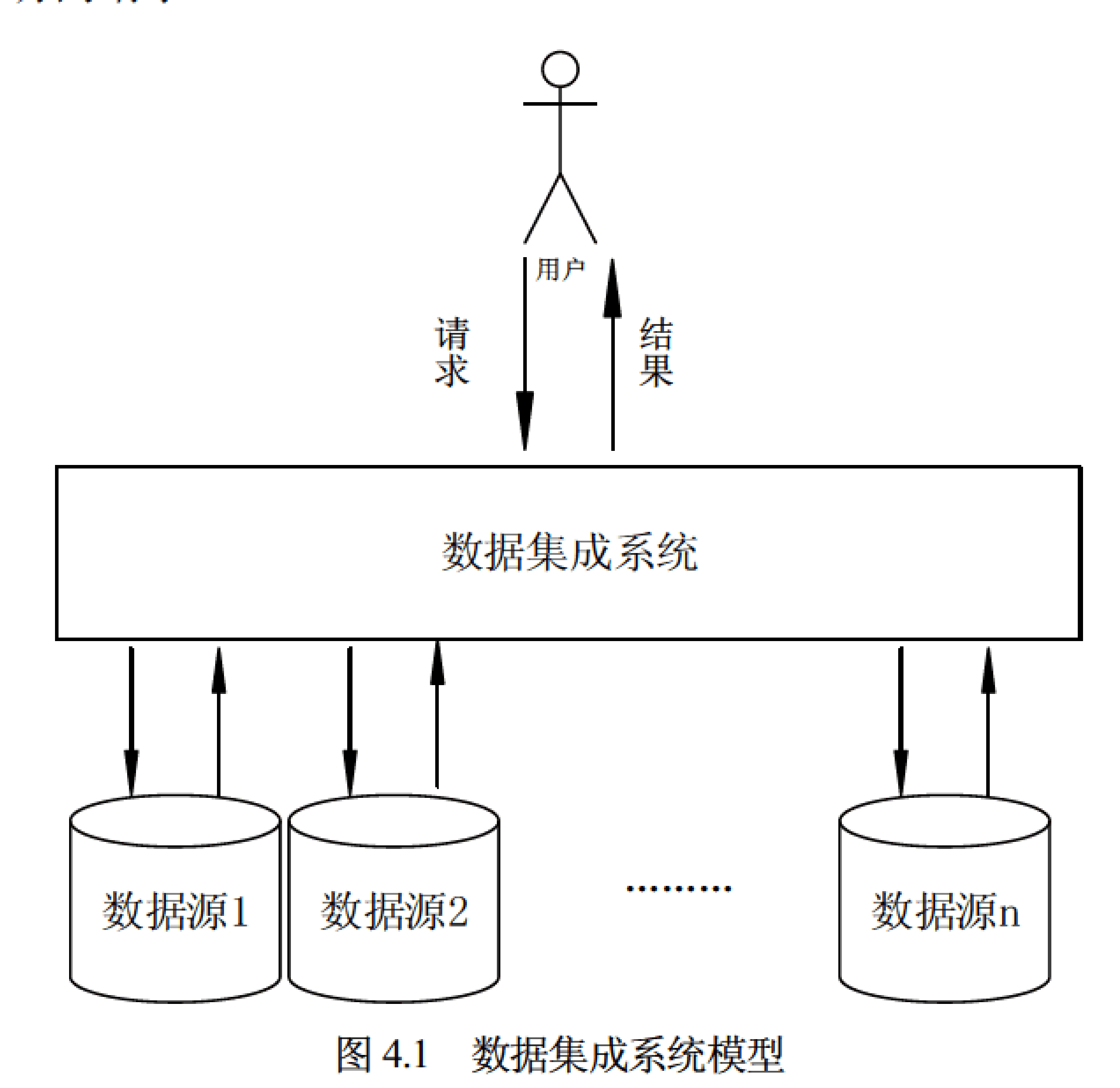


图 4.1　数据集成系统模型

数据集成是对各种异构数据提供统一的表示、存储和管理，这些功能在异构数据集成系统中实现。数据集成屏蔽了各种异构数据间的差异，通过异构数据集成系统进行统一操作。因此集成后的异构数据对用户来说是统一的和无差异的。

数据集成是指将存放在自治和异构数据源中的数据进行组合，向用户提供一个统一的全局数据模式。

一般意义下的数据集成，主要指的是数据库到数据库中异构数据的转换，这种形式的数据集成是其他形式（API，方法等）集成的基础，因为企业应用系统的基础是数据，企业应用集成通常是以数据作为集成的起点。一般情况下，数据集成是必须要实现的。相对其他应用集成的形式而言，数据集成是一种简单、基础的集成方式。主要任务是将数据在存储之间进行转移，在应用程序之间进行业务信息的共享。目前存在大量的工具和技术，允许从数据库到数据库之间的数据集成与转换。相对于应用程序中的逻辑或数据库结构，对数据库的访问和数据转换相对来说比较容易。

在 EAI 的环境下的数据访问运行于应用程序逻辑和用户接口之下，可以通过一个接口从数据库中提取或加载数据。在过去的十几年中，数据库通常与应用程序和接口是分离的，这样就使得

数据集成的难度降低。然而，许多数据库与应用程序的逻辑是紧密耦合的，这样就不能在无视应用程序逻辑的情况下直接处理数据库中的数据。

4.2 数据集成技术研究现状

4.2.1 Information Manifold：具有统一的查询接口

1996 年 Alon Halevy、Anand Rajaraman、Joann Ordille 三人合著的论文《Querying Heterogeneous Information Sources using Source Descriptions》发表在 VLDB 国际会议上，2006 年被评为 VLDB 十年最佳论文。作者在文中总结了数据集成这十几年来的发展成果，在商业领域的一些相关产品，提出了目前数据集成普遍存在的问题以及未来面临的挑战，同时，作者也对数据集成领域中的一些重要思想和几个热点问题进行了详细的介绍。

这篇论文提出了一个数据集成项目——Information Manifold，Information Manifold 和其他同类的项目极大地促进了数据集成的发展，并导致了一系列数据集成系统商业产品的诞生。

Information Manifold 的目的是为多数据源提供一个统一的查询接口。用户通过这个接口提交查询可以直接得到对多个数据源的查询结果，就像是对一个数据源进行查询一样。这个观点深深影响了数据集成领域的发展方向。

请看这个查询的例子：找出由 Woody Allen 导演的在“我”所在的地区放映的电影的评论。

这是一个复杂的查询，要回答这个查询需要对三个 Web 站点（相当于数据库中的表）的内容进行连接：一个有演员和导演信息的电影网站；一个电影放映时间和地点的网站，以及一个影评站点。

如果用户不得不自己访问这三个 Web 站点，然后在三个站点上分别进行有关信息的查询（只能查询该站点的数据库支持的信息），再自己手动把这些信息连接起来，才能得到所需的信息，那么这种复杂度必定是不可忍受的。因此，数据集成研究工作的目标就是设计出一种合适的数据集成系统，它能够自动为用户完成这些操作，并且在可以接受的时间内返回查询的结果数据。至于这些结果信息是否来自多个自治而且异构的数据库、原来的形式是否各不相同等问题，都由系统来解决，用户的感觉就是对单一数据库的简单查询。Information Manifold 就是在这方面比较成功的范例。

Information Manifold 对数据集成这十年来的发展的主要贡献就是论文里提出的对已知的数据源内容的描述方式（称为 Source Description，即源的描述）。一个数据集成系统会给它的用户提供一种模式，用于用户提交他们的查询。其中典型的代表就是中介模式（或称全局模式，mediated schema）。用户提交的查询都是基于这个中介模式的，因此数据集成系统必须预先建立好中介模式与数据源模式之间的语义映射（semantic 映射 s）。在这里，Information Manifold 提出了一种著名的语义映射关系的构建方法，后来被称为 LAV（Local-as-View）方法。有了模式间的映射关系，

用户提交的基于中介模式的查询通过查询重写（query reformulation）转化成对于各数据源的可执行的一系列查询。现在多使用 LAV 视图进行查询重写，被称为利用视图应答查询（Answering Queries Using Views，AQUV）。然后查询引擎再进行查询优化和执行。形象化描述如图 4.2 所示。

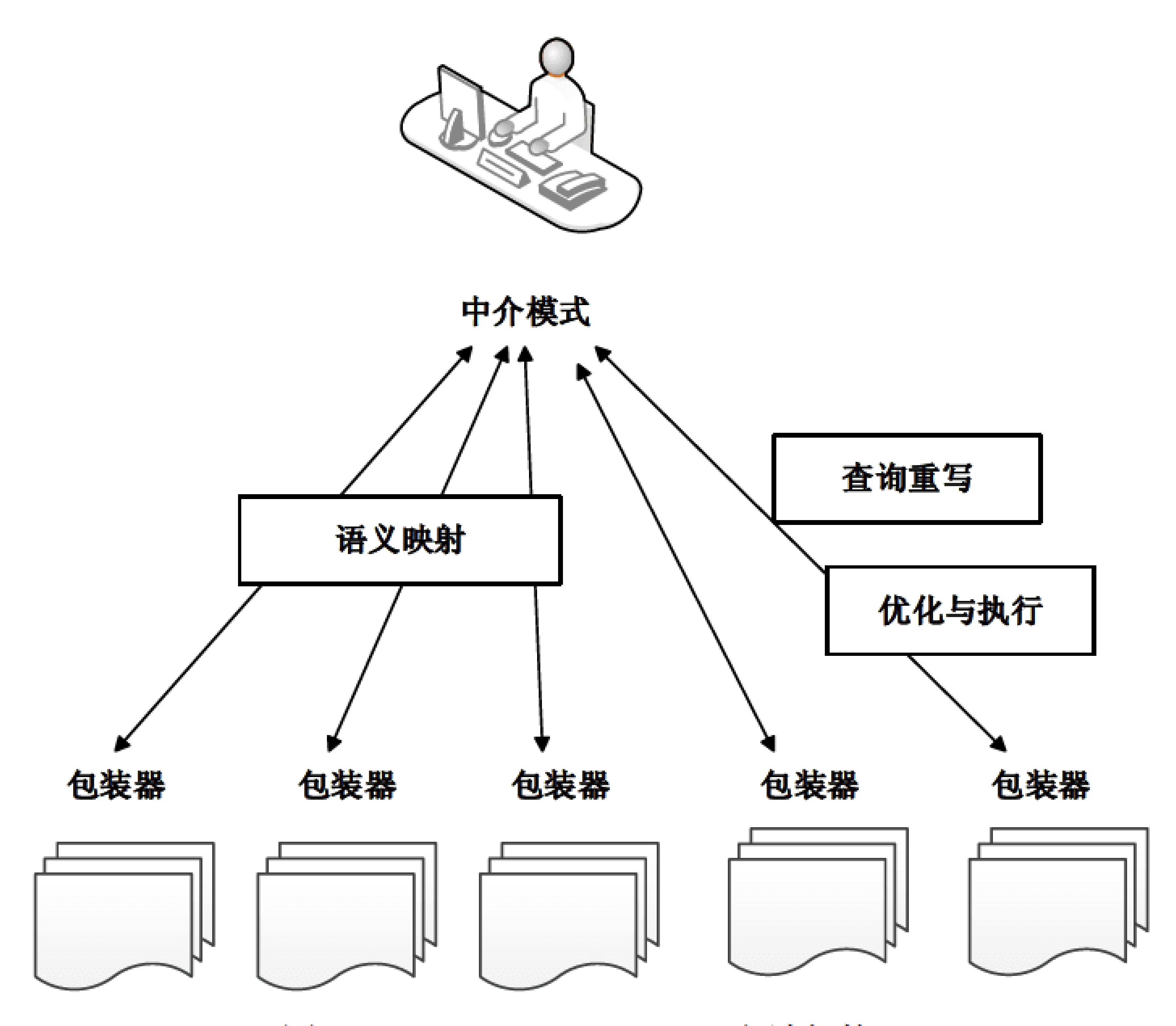


图 4.2　Information Manifold 方法架构

1. 中介模式/全局模式（mediated schema）

中介模式是现在最典型的数据集成方法，它通过提供一个统一的数据逻辑视图来隐藏底层的数据细节，使用户可以把集成的数据源看作一个统一的整体。

数据集成系统通过中介模式将各数据源的数据集成起来，而数据仍存储在各个局部数据源中，通过各数据源的包装器（wrapper）对数据进行转换使之符合中介模式。用户的查询是基于中介模式的，不必知道每个数据源的模式。中介器（mediator）将基于中介模式的一个查询转换为基于各局部数据源模式的一系列查询，交给查询引擎做优化并执行。对每个数据源进行的查询都会返回结果数据，中介器再对这些数据做连接和集成，最后将符合用户查询要求的信息返回给用户。

使用中介模式的数据集成方法解决了各数据源中数据的更新问题。因为当底层数据源发生变化时，只需要修改中介模式的虚拟逻辑视图就可以了，大大减少了数据集成系统的维护开销。

这种方法也弥补了数据仓库方法的不足，数据仓库方法必须将各数据源的所有数据都预先取到一个中心数据仓库里，当数据发生改变时，还要到底层数据源中再取一次，还要更新与这些变化了的数据的相关的那些数据，维护开销太大。

2. 语义映射（semantic 映射）

这里指的是一种能够描述中介模式和数据源模式之间的语义关系的映射，它把多个数据源的模式通过映射关系集成到中介模式上。

这种映射关系就是前面提到的 Source Description 的主要组成部分。

目前，数据集成领域关于模式间映射关系构建的基本方法主要有两种：GAV（Global-as-View）方法和 LAV（Local-as-View）方法。

GAV 方法是将各本地数据源的局部视图映射到全局视图，即全局模式被描述为源模式上的一组视图。用户查询直接作用于定义在数据源模式上的全局视图。GAV 方法的优点是查询效率比较高，缺点是用这种方法构建出来的映射关系的可扩展性较差，不适合数据源存在动态变化的情况。因为一旦有任何一个局部数据源发生改变，全局视图都必须进行修改，维护起来较困难，开销也比较大。GAV 是较早提出的方法。

Information Manifold 提出了一种新的、更适合数据源特点的语义映射关系构建方法，即 LAV 方法。LAV 方法是将全局视图映射到各数据源上的本地局部视图，即各数据源模式被描述为全局模式上的视图。当用户提交某个查询时，中介系统通过整合不同的数据源视图决定如何应答查询。这种方法可看作利用视图回答查询。该方法的优点是映射关系的可扩展性好，适合于信息源变化比较大的情况，缺点是可能会造成“信息遗失”、信息查询效率低。

LAV 方法有如下两个显而易见的好处：

- 第一，描述数据源变得更简单容易了。描述（即视图）只用描述本地数据库就可以了，不必再描述用户查询需要涉及的其他的数据源和各数据源之间的关系。由于有这种特性，当有新的数据源要加入进来时，数据集成系统可以非常容易地适应，因为每个视图仅描述这个数据库的内容。在实际应用的数据集成系统中，往往要涉及成百上千个数据源，而且经常需要去除旧的不用的数据源，加入新的源，再做集成，所以这个容易更新再集成的特性是极其重要的，所以 LAV 方法是现在最流行的数据集成方法。
- 第二，对数据源的描述更加精确了。因为源的描述（Source Description）在视图定义语言的表达能力中起着最关键的作用，因为系统能够选取一个最小数量的数据源集合来回答一个特定的查询，所以比较节省时间和系统开销。

目前兴起的 GLAV（Global-Local-as-View）映射方法是一种 GAV 和 LAV 方法相结合的产物，它是由全局模式上的视图与各数据源上的视图相结合形成的。GLAV 方法可以结合 GAV 和 LAV 的优势，能够为数据集成系统提供更具表达能力的语义映射。

3. 查询重写（Query Reformulation）

数据集成系统为多数据源提供统一的接口，利用视图描述一个自治的、异构的数据源的集合。用户基于中介模式提交一个查询，数据集成系统通过源模式与中介模式之间的映射关系将该查询重写为数据源可接受的语法形式传给数据源，在随后的阶段基于数据源的查询被优化并执行。

4. 利用视图应答查询（Answering Queries Using Views，简称 AQUV）

AQUV 也被称为利用视图重写查询（Rewriting Queries Using Views），即给定一个数据库模式上的查询 Q，和同一数据库模式上的视图定义集 V={V1，V2，…，Vn}，能否仅使用视图 V1，V2，…，Vn 获得对查询 Q 的应答。

在使用 LAV 方法构建映射关系的数据集成系统中，各数据源模式是全局模式上的视图，数据源的内容由在中介模式上的视图来描述。因此可以将数据源看成是物化的视图（Materialized Views），将视图定义看成是数据源描述（Source Description）。从而将在中介模式上构造的用户查询，重写为一系列的直接基于各数据源模式的查询，这就是利用视图应答查询问题。

有时候不一定能得到与用户查询等价的重写查询，原因是物化视图越来越多，想全部覆盖这些视图是很困难的。在有些情况下，作为近似，可以找到最大包含集，它提供可用数据源上可能的最佳结果集。

因此查询重写分为两种类型：

- 相等的查询重写：重写的查询与原查询有相同的结果集，可以理解为等价的查询重写；
- 最大包含的查询重写：重写的查询是原查询的最大子集。

4.2.2 数据集成系统的发展建设

1. 模式间映射关系的生成

模式和模式间的语义映射关系是数据集成系统的构建基础。

现在，建立 Source Description 已经迅速成为开发实际应用的数据集成系统的最主要的瓶颈。更准确地说，瓶颈是建立源模式与中介模式之间的语义映射关系。要创建这样的映射关系并且维护它们，需要专门的数据库专家来完成，而且，他们还必须同时具备丰富的商业知识，才能够理解需要进行匹配的模式所具有的意义。对于企业来说，聘请这样的专门人才来建立和维护数据库的模式匹配关系，代价肯定是比较大的。

有需求就有发展的动力。这促成了数据集成研究领域里的一个相当重要的分支：半自动化生成模式映射关系。一般来说，完全自动地生成映射关系是一个几乎不可能完成的问题，因此研究努力的方向应该是创造出能够加速映射的生成并且尽可能减少人工干预的工具。

在自动化生成模式匹配的研究领域中，现有的工作都是基于这样的思想：第一，用于建立模式之间的匹配的技术都基于那些模式本身所包含的线索，比如模式元素与数据值或属性值在语言上的相似性与重叠性。第二，据观察，这些方法没有一个是十分简单的，以后的数据集成系统的发展趋势必然是联合这一系列单独的技术，来创建模式之间的映射关系，才能达到比较良好的效果。第三，一个重要的观察结果是，模式匹配的创建工作常常具有很大的重复性。例如，在做数据集成时，我们建立同一个域上的多个模式到同一个中介模式的映射关系。因此，可以使用机器学习算法，这种方法是：先人工建立一个初步的模式映射关系，作为训练数据，然后对这些映射做归纳，预言产生出其他那些未知的模式间的映射关系（见图 4.3）。这些技术今天已经在商业领域中使用，并且带来了重要的商业价值。

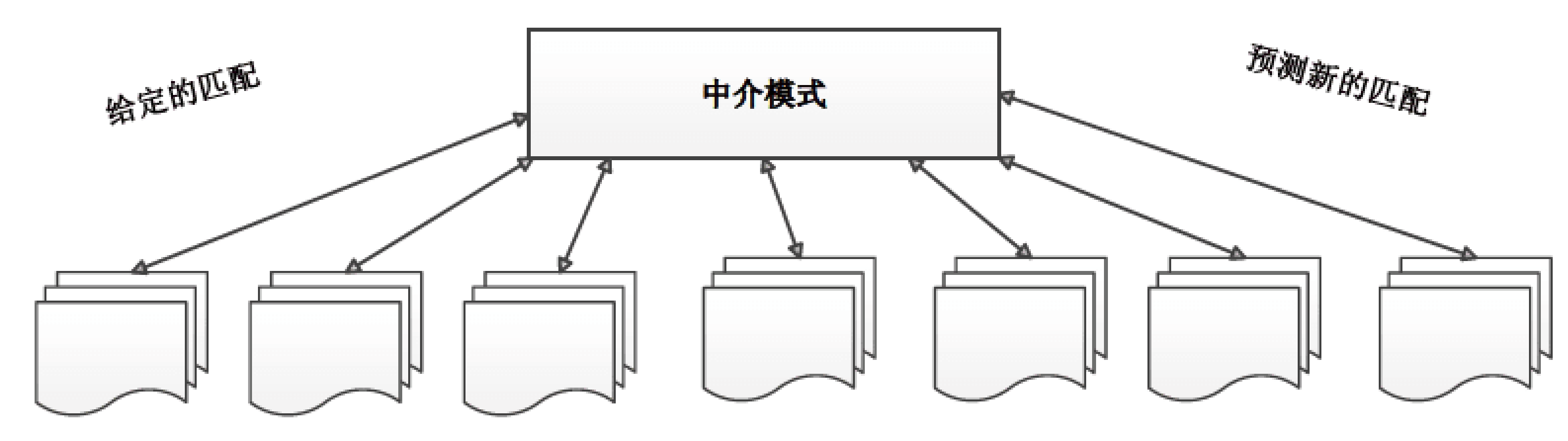


图 4.3　中介模式：基于机器学习的方法

2. 适应性查询处理

一旦一个被提交给中介模式的查询被重写为一系列的面向各个数据源的查询，这些查询就需要被有效率地执行。尽管分布式数据管理中有许多技术在这里都很适用，但又有一些新的挑战出现了，主要是由于数据集成系统中的信息的动态特性决定的。

数据集成系统与传统的数据库系统不同，它的各个数据源具有自治性和异构性，各个数据源数据的可访问性以及传输速度是经常变化和不可预测的，执行引擎没有足够的信息来制定出一个好的查询计划。因此传统的停止-进行方式的查询处理不能很好地处理数据集成系统的查询。而能够在查询执行过程中动态调整查询计划的适应性查询处理是针对此类应用的最佳选择。适应性查询处理逐渐成为一项重要的技术。

3. XML

我们不能忽视 XML 在过去十年的数据集成发展史上所起的重要作用。

如今的 Web 数据库实质上就是一个巨大的异构数据库的集合，怎样为大量异构的数据提供某种统一的表示方法无疑是数据集成研究领域中的重要问题。这就要求我们找到一种标准的、开放的数据结构来表示数据。而 XML 的出现无疑为异构数据源的集成带来了新的希望。

XML 是互联网联合组织（W3C）设计并推荐的新一代可扩展标记语言，它是 SGML 的一个优化子集。它以一种开放的自我描述方式定义数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系。XML 是一种半结构化的数据模型，它的很多特性使得它可以描述不规则的数据，能够集成来自不同数据源的数据，可以将多个应用程序所生成的数据纳入同一个 XML 文件。

实际上，XML 没有解决任何语义集成的问题，那些数据源共享 XML 文件，然而这些文件的标签在这种应用之外就是毫无意义的。可是，用户看起来的效果是好像这些数据源里的数据真的被共享了一样，而且用户的操作也像是在一个真正的、数据共享的数据集成系统中进行的一样。现在 XML 对数据集成研究的推动力越来越重要了。

如果没有 XML，集成系统就必须了解每个数据库描述数据的模式和规则，这几乎是不可能实现的。Web 数据源中的数据表示形式的不同几乎是无穷无尽的，XML 能够使不同来源的结构化的数据很容易地结合在一起。

从技术的角度来看，目前一些数据集成系统已经使用了 XML 作为它的基本数据模型，并且用 XML 查询语言（XQuery）作为数据库查询语言。要维护和支持这样的系统，数据集成系统的

每一个方面都需要被扩展，使之具有支持和处理 XML 的能力。

主要的挑战是 XML 的嵌套特性，而且 XML 是半结构化的语言。

4. P2P 数据管理

点对点（Peer-to-Peer）文件共享系统的兴起，鼓舞了数据管理研究领域对 P2P 结构实现数据共享的兴趣。除了 P2P 模式的常规要求以外，研究者们还提供了 P2P 在数据集成环境下的两种附加的优点。

第一，在实际应用中，几个不同的组织要求共享数据，这种情况经常发生。但是这些组织中却没有一个想要担负起创建一个中介模式、维护它，并且为它建立和那些数据源模式之间的映射关系的责任。这怎么办呢？P2P 结构为我们提供了一个非常好的解决办法。P2P 结构提供的是一种真正的分布式管理共享数据的模式，每一个数据源仅仅需要提供它自己与它周围一系列邻居数据源的语义映射关系，其他更复杂的集成是系统依循着网络中的语义路（Semantic Paths）形成的。源的描述（Source Description）提供了研究 P2P 结构下的模式及其映射的建立的基础。

第二，设计一个单独的中介模式为一个数据集成系统服务，有时候会比较难，而且一个单独的中介模式又比较难以将系统中全部的语义关系都表示清楚。请考虑一个科研合作环境下的数据共享问题，需要被共享的数据可能包括来自不同大学的科研成果，不同书籍上的信息，等等。数据的多样性和异构性，以及合作团体对于共享这些数据的需要，都是非常多样而且经常变化的，这些特性对于一个单独的中介模式来说，都是极其难以管理的。但是 P2P 模式就不同了，在这种结构下，没有一个单独的全局的中介模式，数据的共享只发生在网络上这个数据源的邻居数据源之间。

图 4.4 所示为 P2P 模式的一个形象化示意。

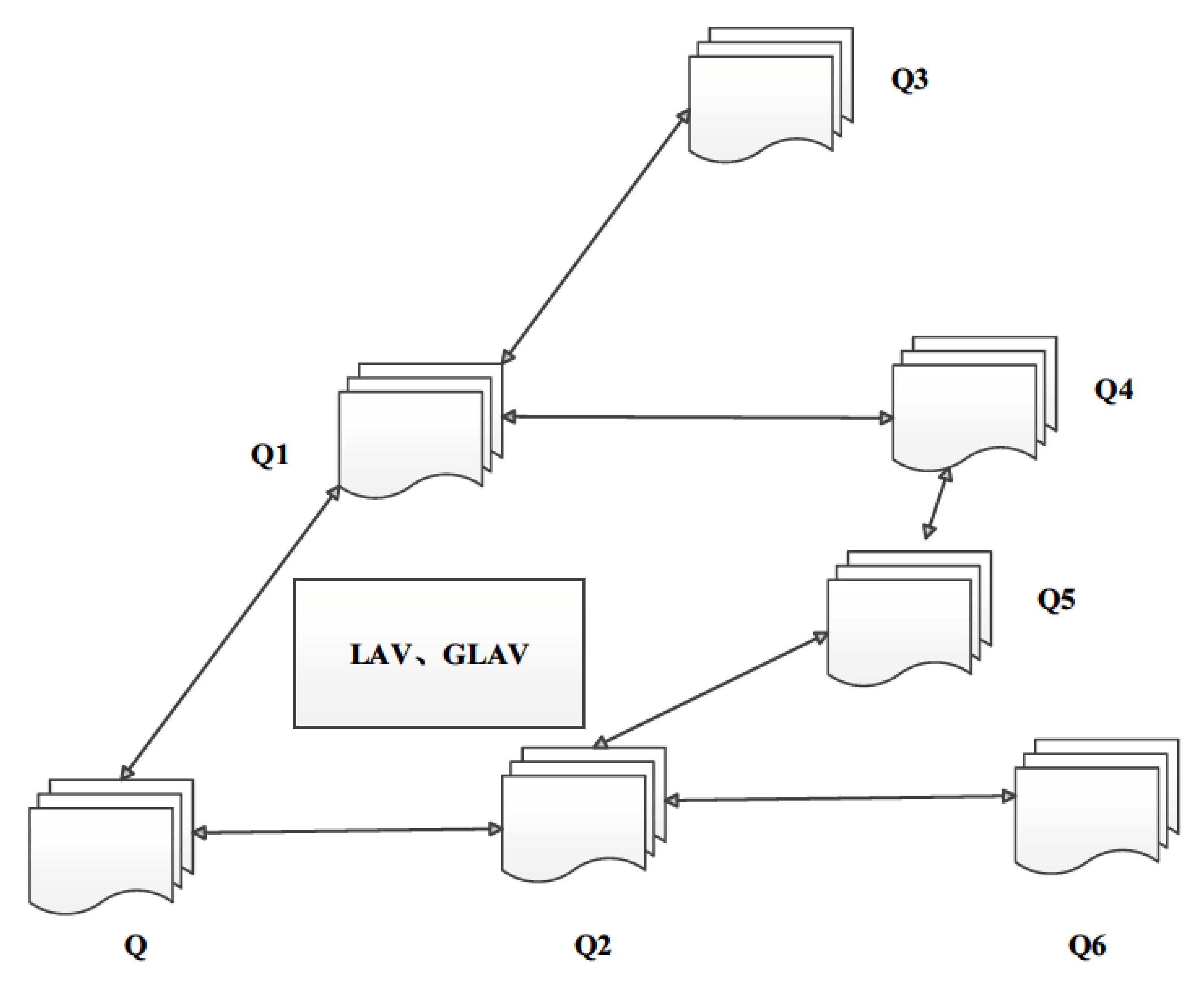


图 4.4　P2P 数据管理示意

5. 人工智能的重要作用

数据集成在人工智能（AI）的领域里也是一个非常活跃的研究课题。在早期，数据集成在人工智能领域的应用被称为描述逻辑（Description Logics），它是知识表示的一个分支，能够描述数据源之间的关系。Information Manifold 系统的中介模式就是基于典型的描述逻辑，它把描述逻辑的表达能力同数据库查询语言联合起来了。描述逻辑为中介模式的表示，还为语义查询的优化提供了更加灵活的机制。

机器学习在为数据集成系统半自动化地建立语义映射这个领域扮演了一个非常重要的角色。可以预言，未来机器学习将会对数据集成有着越来越重要的影响。

4.2.3　企业信息集成

20 世纪 90 年代末开始，数据集成从实验室里面“走”了出来，进入到了商业化领域中，成为现代化企业信息管理必不可少的应用技术。今天，这种工业被称为企业信息集成（Enterprise Information Integration，EII）。

现代企业对于数据集成的需求日益增长，试图找到一种用单一系统对企业的所有信息资产实现集成和管理的解决方案，从而达到有效地集成企业信息，对多个数据库统一管理的目的。

EII 工具的出现解决了数据管理领域的一个非常让人头痛的问题——从多个数据源提取数据。它的根本思想是：为来自多个不同数据源的信息提供集成工具，这种工具无需首先把所有的数据从网上下载到本地的数据仓库里。这正是 EII 工具的优越和先进之处。

EII 系统中，数据是“随需应变”地抽取。查询经过优化、分段又被返回所有的数据源，而结果则被放入到数据源的虚拟视图，“虚拟”是从数据通常都是驻留在数据源的意义上来说的。EII 工具是“访问”而不是“移动”数据。这就从根本上简化了分布数据的访问和集成。

和任何新兴的产业一样，EII 也面临着许多挑战，下面是具有代表性的一些。

1. 水平 vs 垂直

从商业角度来看，EII 公司必须决定是要建造一个能在任何应用环境下使用的水平平台，还是为某一个特殊的垂直方向制造特定的工具。这就是 EII 发展中的水平 vs.垂直（Horizontal vs. Vertical）问题。

- 垂直方法的观点是：用户更关心他们的全部问题能否都被解决，因此在解决方法中必定有一个“纵深”方向很适合解决某个用户的问题，所以我们要向下深入研究这个方面，不必特别关心解决方法中的其他方面，以及它与解决方法的其他方面的整合。
- 水平方法的观点是：系统的一般性使人很难断定哪一个“垂直”的方向是我们在解决方法里要特别关注的。所以建立一个通用性较好的“水平”平台更重要。

对于一个新建立的公司来说，这是一个在现有资源不足的情况下如何区分建设的优先次序的热点问题。

2. 和 EAI 工具以及其他一些中间件的整合

数据管理的中间件产品是一个非常复杂的问题，EII 工具的出现又把这种复杂度加剧了。一个更为成熟的工具是企业应用集成（Enterprise Application Integration，EAI），它可以通过中间件作为粘合剂来连接企业内外各种业务相关的异构系统、应用以及数据源。

EAI 的核心就是使用中间件连接企业应用，使应用更加便利，EII 则更关注于集成数据和查询。然而，从某种意义上来说，数据是为了应用服务的，查询得到的数据是要放入到其他的数据源中。事实上，要查询数据，最好使用 EII 工具；但是若要更新数据，那么就必须求助于 EAI 工具。因此，EII 和 EAI 工具的分离也许只是一个暂时性的问题。其他的产品包括数据清洗工具（data cleaning tools）和记录分析工具（reporting and analysis tools），这些工具与 EII 和 EAI 的结合将会有重大的进步。

尽管面临着这些挑战，还有激烈的竞争和因特网泡沫破裂后极其困难的商业环境，EII 产业仍然存活了下来，今天它已经成为现代企业的一项不可缺少的技术。

除企业市场之外，数据集成在因特网搜索研究领域里也扮演着相当重要的角色。到 2006 年，大型的搜索公司（比如 Google）在集成来自 Web 上的多个数据源中的信息方面，取得了一定的进步。在这里，源的描述（Source Description）起了至关重要的作用：因为给无关数据源发送的巨大的查询量的开销是非常高的。因此数据源必须要被尽可能精确地描述。而且，垂直搜索（vertical search）关注于创造特殊的搜索引擎，集成来自于某一特定领域（如旅行、工作等）的多个 Deep Web 数据源上的数据。垂直搜索引擎产生于 Web 的早期（比如 Junglee and Netbot 公司）。这些搜索引擎中也包含了复杂的源描述。

4.2.4 未来的挑战

几个基本因素决定了数据集成研究将面临着长期的挑战。

- 第一个因素是社会性的。数据集成的本质是人们合作和共享数据的问题。它包括找到合适的数据，使数据集成系统的用户相信这些数据的来源、正确性和安全性，并愿意共享它们。（这需要考虑到用户的想法，他们愿意共享这些数据可能是因为共享数据的便利性或是应用结果带来的好处。）还要使数据的拥有者相信，他们所有的关于共享数据的担心，包括私密性、系统的查询性能表现等，都会被妥善解决。
- 第二个因素是集成的复杂性。在很多应用环境下，人们并不清楚“数据集成”的意义是什么，也不知道如何对已经联合在一起的一堆数据进行操作。数据管理系统的设计者必须考虑到这种情况：用户的要求有时可能会导致这种预料不到的数据集成的复杂性。系统必须能够适应这种状况。

由于以上这些原因，数据集成被认为是一个和人工智能一样难的问题，甚至更难。因此，研究者们的目标应该是以多种方案、不同角度创造能够使数据集成变得更加便利的工具。

以下是几个目前比较流行的数据集成领域的创新与挑战。

1. 数据空间（DataSpaces）：Pay-as-you-go 的数据管理模式

现在的数据库系统和数据集成系统的一个基本的缺点是：需要很长的建立时间。创建一个数据库系统，必须首先建立一个模式，然后向数据库中增添元组。等这些工作都完成以后，才能够给用户提供查询服务。创建一个数据集成系统，需要预先建立中介模式到数据源模式之间的语义关系，才能看得到数据源中的内容。近年来，一种新的数据管理模式——数据空间出现了，它强调的是一种 pay-as-you-go 的数据管理模式：不需要任何的建立时间就能够给用户提供服务。随着时间的推移，用户的需求不断增加，数据空间系统“增量式”地添加服务的内容，改进服务的质量，这个过程也是数据不断被集成的过程。因此数据空间并不像数据集成系统那样，先把数据集成好了，再给用户提供服务，而是“随需要随集成”的方法，即上面提到的 pay-as-you-go 方式，如图 4.5 所示。

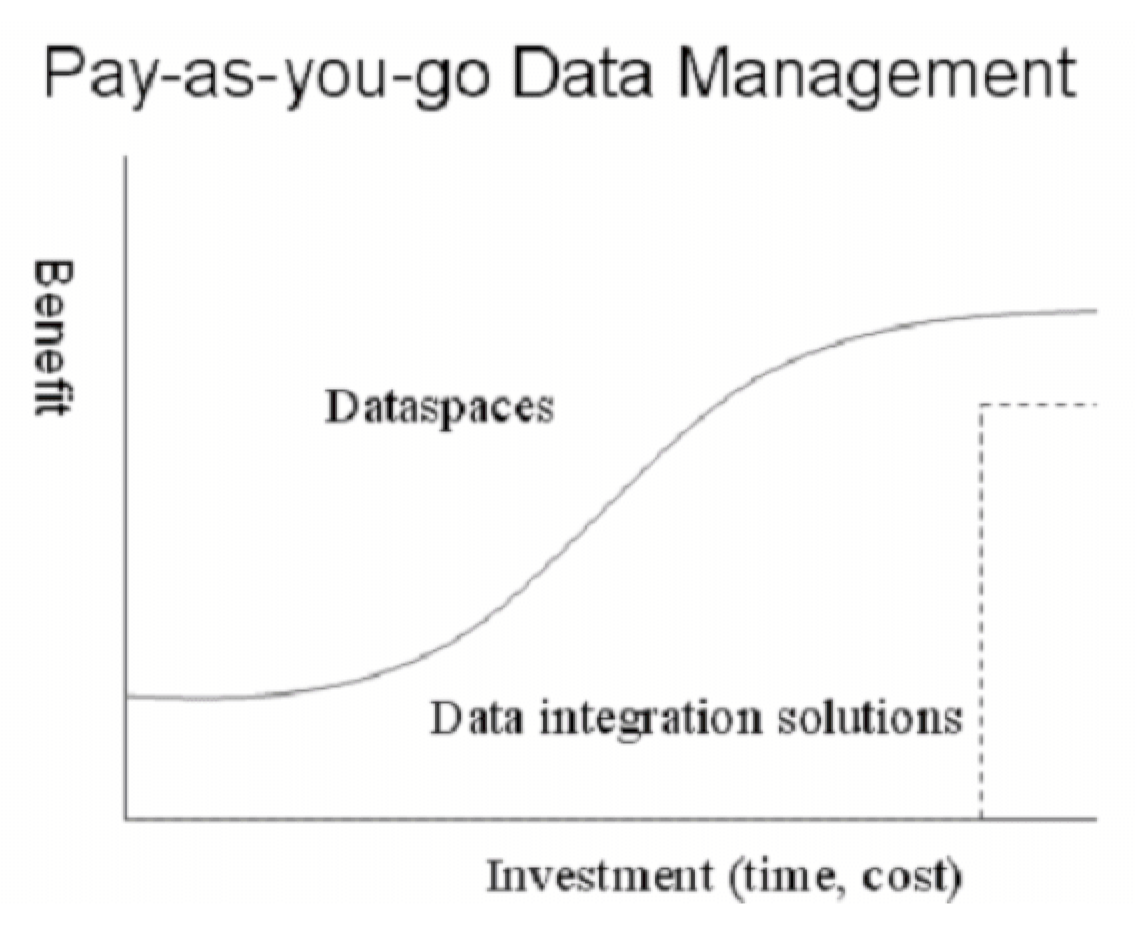


图 4.5　随需数据管理方案

在数据空间的最早期，只能提供一些最基本的（如数据源上的关键字查询之类）功能。数据空间使用一系列启发式的抽取规则，从本来完全互异的、毫无联系的数据项中析取出它们之间的关系，使用路径查询方法建立这些关联。最终，当两个数据源之间确实需要更紧密的集成时，数据空间就可以自动创建两者之间的映射。接下来的事情就是让人去修改并维护它了。

2. 不确定性与数据血统（Uncertainty and lineage）

在数据集成研究领域，不确定数据的操作和数据血统的问题有很长的历史。如果说管理不确定性数据和数据血统在传统数据库系统中似乎只是一个好的特点，那么在数据集成系统中它就是一个必须具备的功能了。一般情况下，来自于多个数据源的数据都是不确定性的数据，它们彼此的形式都不一致。系统必须能够找出这些看似乱七八糟的数据中内在的联系和确定性。当系统不能自动找出这种确定性的时候，可以交由用户来考虑一下数据的血统（也叫数据沿袭），搜索引擎沿着用户的搜索过程把这些 URL 都提供给用户，因此用户能够通过分析 URL 理清数据的脉络，决定哪个搜索结果更值得深入探寻下去。通过对数据血统的分析，用户可以知道数据何时更新、如何计算以及从何处而来，这些帮助用户追溯数据产生的来源。这种深入洞察数据来龙去脉的能

力能够帮助用户断定哪个数据源是可信赖的。

3. 重新使用人们的关注点（Reusing human attention）

若要在数据源上做更加紧密的语义集成，一个重要的原则就是：要重新利用用户的关注信息。一个简单而明显的例子就是，每一次用户使用数据空间系统进行查询，数据空间都能从中得到一条用户关注信息的语义线索。这样的线索可以从用户查询数据源时得到。若用户建立语义映射，或者剪切数据，再把它粘贴到另一个地方，这些操作都能给系统提供很多用户关注点的信息。如果能够建立一个支持这些语义线索的系统，那么语义集成将会变得非常快。目前已经有了一些重用用户关注信息的成功案例。

4.3 数据集成技术的实现与工具

4.3.1 Oracle Data Integrator（ODI）简介

ODI 的前身是 Sunopsis 公司的 Sunopsis Active Integration Platform，被 Oracle 公司于 2006 年 10 月收购，同时命名为 Oracle Data Integrator。ODI 体系结构内含多个组件，主要围绕一种模块化存储库进行组织。组件主要有图形模块、运行时组件、存储库及 Metadata Navigator 等。

Oracle Data Integrator 在多项组件上得到构建，且这些组件围绕同一集中式元数据库共同工作。图形模块、运行时组件以及 Web 界面，通过与其他高级特性的结合，使 Oracle Data Integrator 成为一种代表当前最先进技术水平、非传统性（Legacy-free）的轻量级集成平台。

Oracle Data Integrator 体系结构的组织主要是围绕一种模块化存储库展开，通过完全在 Java 中编写的图形模块和执行代理（程序）、以客户服务器模式对该存储库进行访问。该体系结构还包括 Web 应用以及 Metadata Navigator，它们可以使用户通过 Web 界面访问信息。

1. 图形模块

ODI 中共拥有 4 种图形化用户模块，分别是：设计器（Designer）、操作器（Operator）、拓扑管理器（Topology Manager）和安全管理器（Security Manager），如图 4.6 所示。这 4 种模块可以被安装在所有支持 Java Virtual Machine 1.5（J2SE），包括 Windows、Linux、HP-UX、Solaris、AIX 以及 Mac OS 在内的图形平台上。

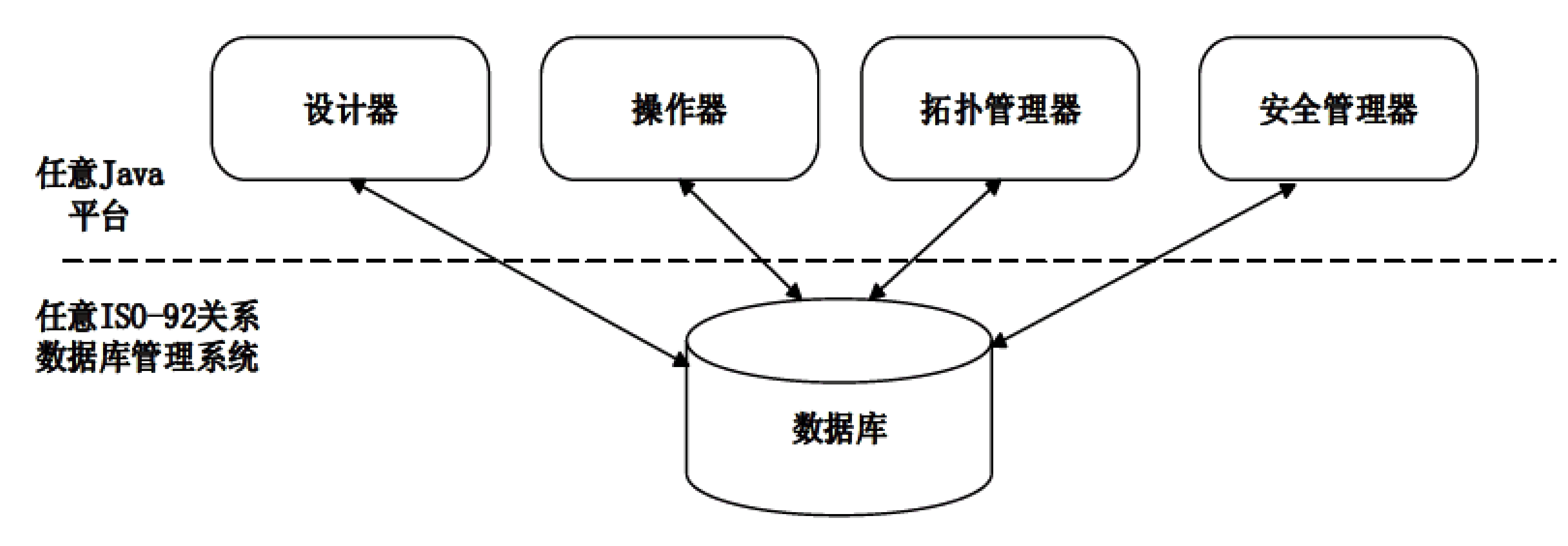


图 4.6　ODI 中 4 种图形模型

（1）设计器（Designer）

这个工具能够让用户为数据转换和数据完整性定义声明式规则。此外，我们能够看到，数据库和应用系统的元数据能够在此模块中被导入和定义。设计器模块使用元数据和规则为实际的生产环境生成方案。所有项目的开发都可以通过这个界面完成，并且它还是开发人员和元数据管理员在设计阶段的主要用户工具。

（2）操作器（Operator）

用户可以利用操作器在生产环境中负责管理和监控运行工作。它的主要使用者是生产环境的操作者，能够显示执行日志，包括错误记录、被处理的行数、执行的统计信息和被执行的实际代码等。开发人员在设计阶段也可以通过 Operator 模块进行调试应用，Operator 已成为运行阶段的核心工具。

（3）拓扑管理器（Topology Manager）

用户利用 Topology Manager 工具来定义基础架构的物理和逻辑体系结构。通过该模块，使服务器、方案和代理（程序）被注册在 Oracle Data Integrator 主数据库中。它主要由基础架构负责人或项目管理员使用并执行相关工作。

（4）安全管理器（Security Manager）

安全管理器的主要使用者是安全管理员。它能够让管理员对用户的账户密码及访问权限等进行有效管理，还可以给 Oracle Data Integrator 对象和功能分配相关的属性信息和用户访问权限。

以上 4 种模块均在集中式存储库中进行信息存储。

2. 运行时组件

在运行时，Scheduler Agent 负责协调场景的执行。它也可以被安装在所有支持 Java Virtual Machine （J2SE）的平台上。其作用仅仅是完成从执行存储库中获得代码，然后向数据库服务器、操作系统或脚本引擎发出执行该代码的请求工作。执行完成后，Scheduler Agent 开始更新数据库中的执行日志，如出现错误可以及时报告错误信息以及执行统计。用户则可以通过操作器 Operator 模块或 Metadata Navigator 的 Web 界面浏览执行日志信息，如图 4.7 所示。

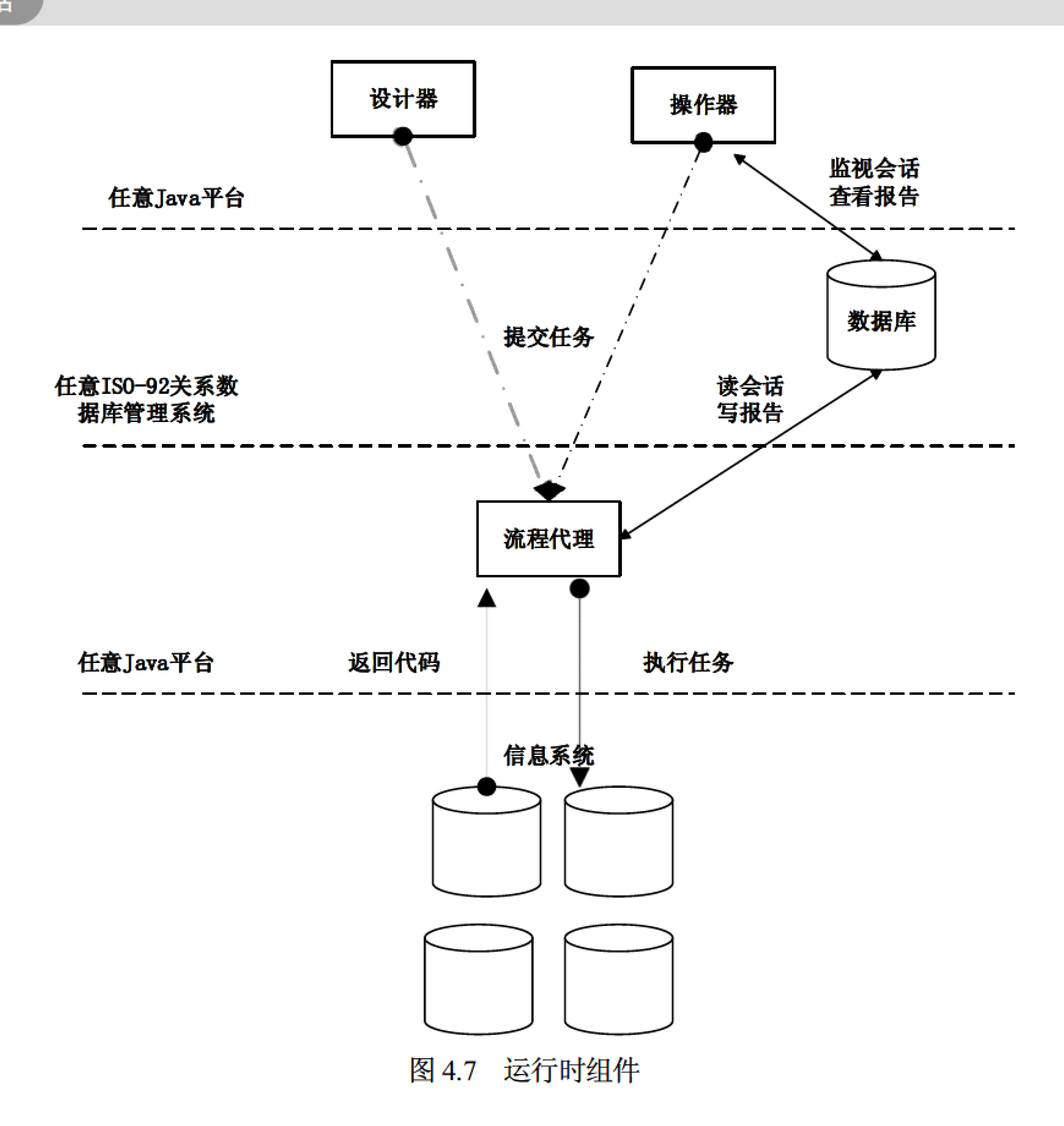


图 4.7　运行时组件

3. 存储库

存储库由一个主存储库和几个工作存储库组成。这些存储库是存储在关系数据库管理系统中的数据库。模块配置、开发或使用的全部对象被存储在其中的一个存储库中，并且通过各种体系结构的组件，以客户服务器模式得到访问。

通常，存储库中仅有一个主存储库，其中包含安全信息（用户资料及权限）、拓扑信息（技术及服务的定义）以及目标的各版本。利用 Topology Manager 以及 Security Manager 使包含在主存储库中的信息得到保留。由于上述模块中均存储拓扑和安全信息，所以，它们都具备对主要存储库的访问权限。这些信息之所以能够都得到安全的保存，都是依靠拓扑管理器和安全管理器来实现。工作存储库主要存储一些项目的信息：模块配置、项目开发或使用的全部对象及运行时信息，包括数据存储、列、数据完整性约束、交叉索引、声明式规则、软件包、程序、场景、日志等。用户可以利用 Designer 和 Operator 模块对工作存储库里的内容进行管理，图 4.8 所示为存储库架构图。

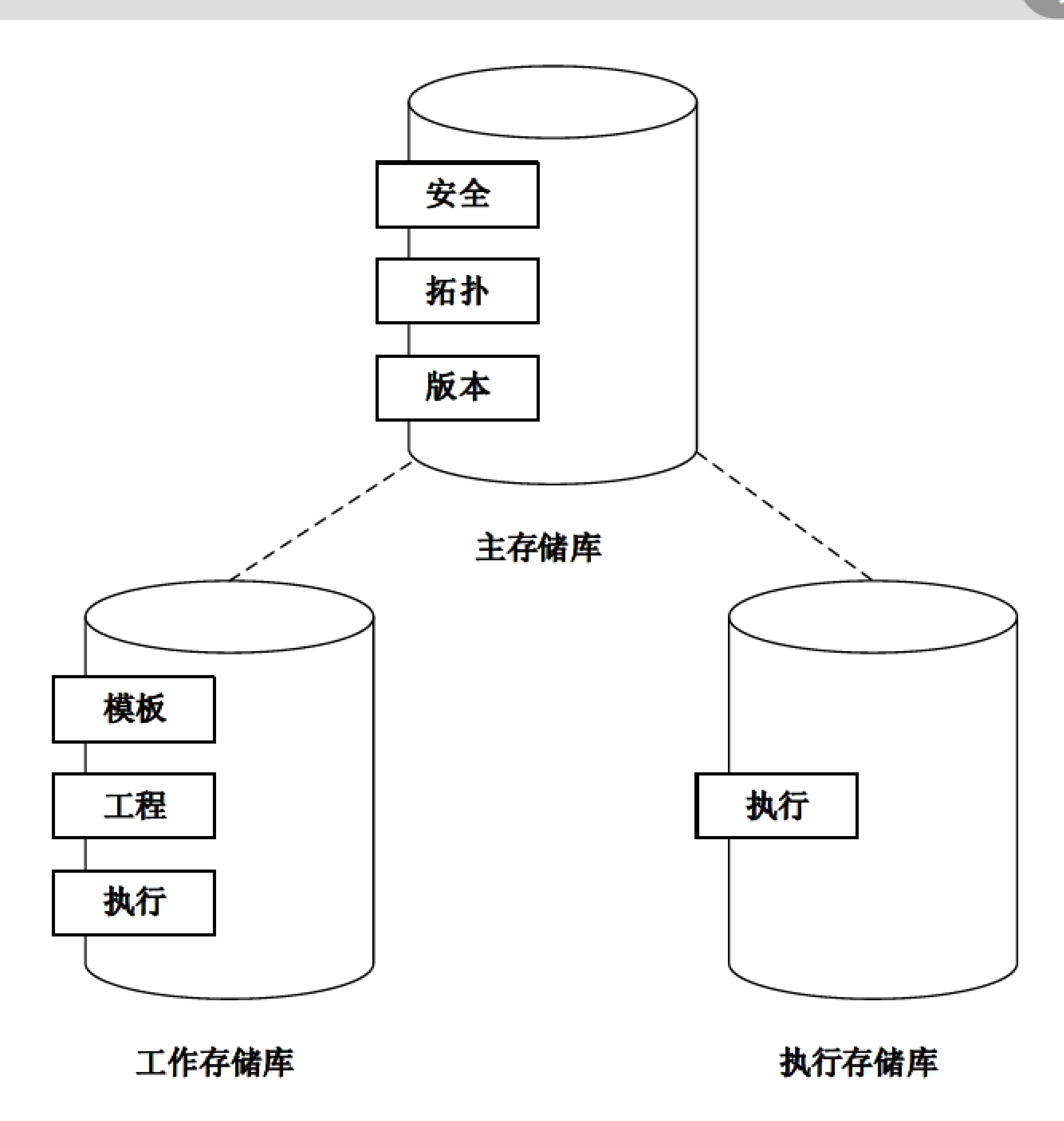


图 4.8　主存储库及工作存储库

用户利用 Designer 以及 Operator 模块对工作存储库的内容进行管理。还可以通过运行时的 Agent（代理程序）对工作存储库进行访问。在工作存储库仅被用于存储执行信息时（通常出于生产目的），可以将其称为执行存储库。在运行时，利用 Operator 界面以及通过 Agent（代理程序）可以对执行存储库进行访问。然而，所有工作存储库始终附属于一个并且是唯一一个主存储库。

4. Metadata Navigator

Metadata Navigator 是一种 Java 2 Enterprise Edition（J2EE）应用，可以提供对存储库的 Web 访问。它使得用户能够对对象进行浏览，包括项目、模块以及执行日志。Metadata Navigator 可以被安装在应用服务器上，诸如 Oracle Container for Java（OC4J）或 Apache Tomcat。

企业用户、开发人员、操作人员以及管理人员可以通过 Web 浏览器访问 Metadata Navigator。凭借其全面的 Web 界面，用户能够看到流程图、追踪所有数据源，甚至深入到字段级了解构建数据所使用的转换。用户还可以通过 Metadata Navigator 从 Web 浏览器中发布和监控场景，如图 4.9 所示。

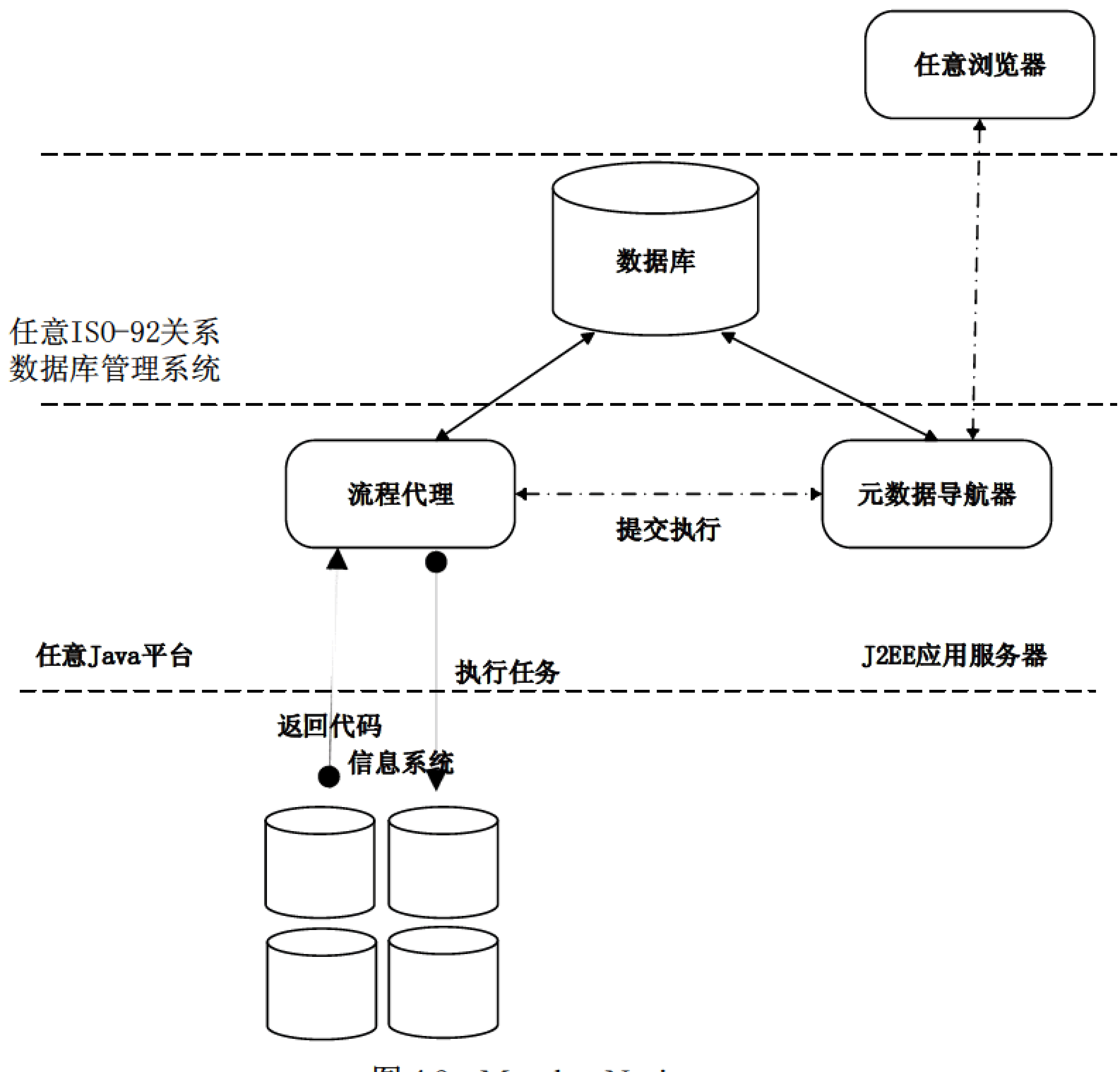


图 4.9　Metadata Navigator

5. 其他组件及特征

Oracle Data Integrator 还包括以下可选择性组件及特征：

- 知识模块可以使技术、数据库以及应用程序快速便捷集成的实现成为可能。它们存在于范围广泛的平台中，包括 Oracle、Teradata、Sybase IQ、Netezza、SAP/R3、Oracle Applications、Siebel、LDAP 以及 XML。
- 具备负载均衡的 Advanced Parallel Option 特性，通过自动平衡几种 Agent（代理程序）间的工作负载，使数据的大批量处理成为可能。
- 高级版本管理可以提供一种管理、保护以及复制工作单元修订的界面，即使是在最大程度的开发环境下。
- Common Format Designer（CFD）特性使用户能够设计或从其他数据模型中快速组装数据模型，并随后自动生成用于加载和从该模型中提取数据的流程。诸如，用户能够利用 Common Format Designer 通过集合异构源生成操作性数据存储、数据中心或数据规范格式。它还可以被用来设计数据仓库模型（诸如，Star 或 Snowflake Schema、3NF）。

Oracle Data Integrator 体系结构的组织使之成为一个代表着当前最先进技术水平的较为完整的数据集成平台，可以满足近乎所有数据集成的需求，其界面如图 4.10 所示。

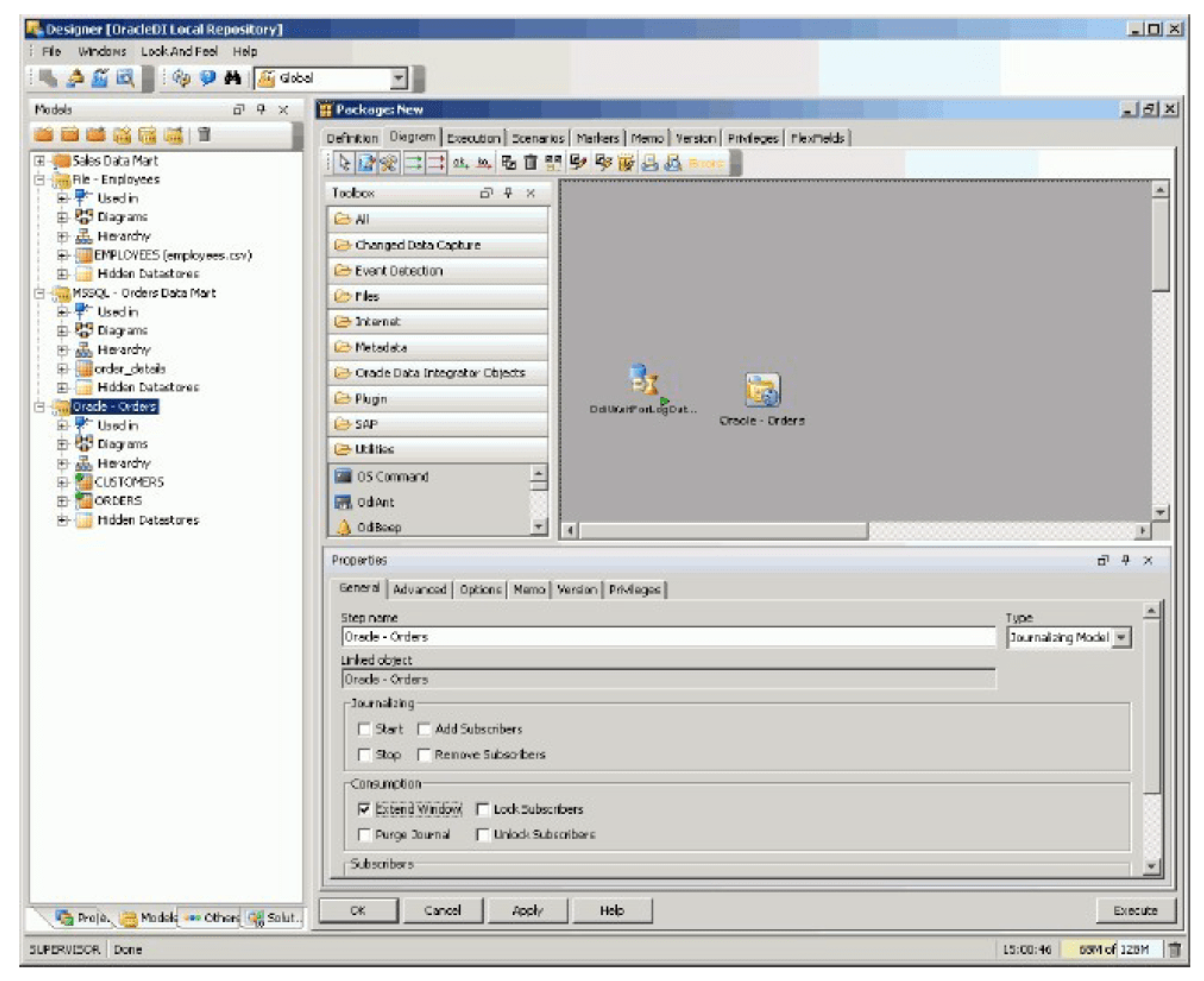


图 4.10　ODI 界面

ODI 主要在 ETL 和数据集成的场景里使用，并提出了知识模块的概念，把场景中详细的实现过程作为一个一个的知识模块，利用 Python 脚本语言并与数据库的 SQL 语句相结合，录制成详细的步骤正确地记录下来，因此在 ODI 中就形成了 100 多个知识模块，基本上包含了所有一般应用所涉及的所有场景。在 ODI 中将这 100 多个知识模块进行了归类，大致上可以分为以下 6 大类：反向工程（RKM）、正在加载（LKM）、检查（CKM）、集成（IKM）、日记（JKM）和服务（SKM）（图 4.11 所示为 Oracle Data Integrator 中知识模块的体系结构图）。知识模块是 Oracle Data Integrator 的核心部分，能够实施真正的数据流，并定义每个流程在多个系统上生成代码所有的模块。

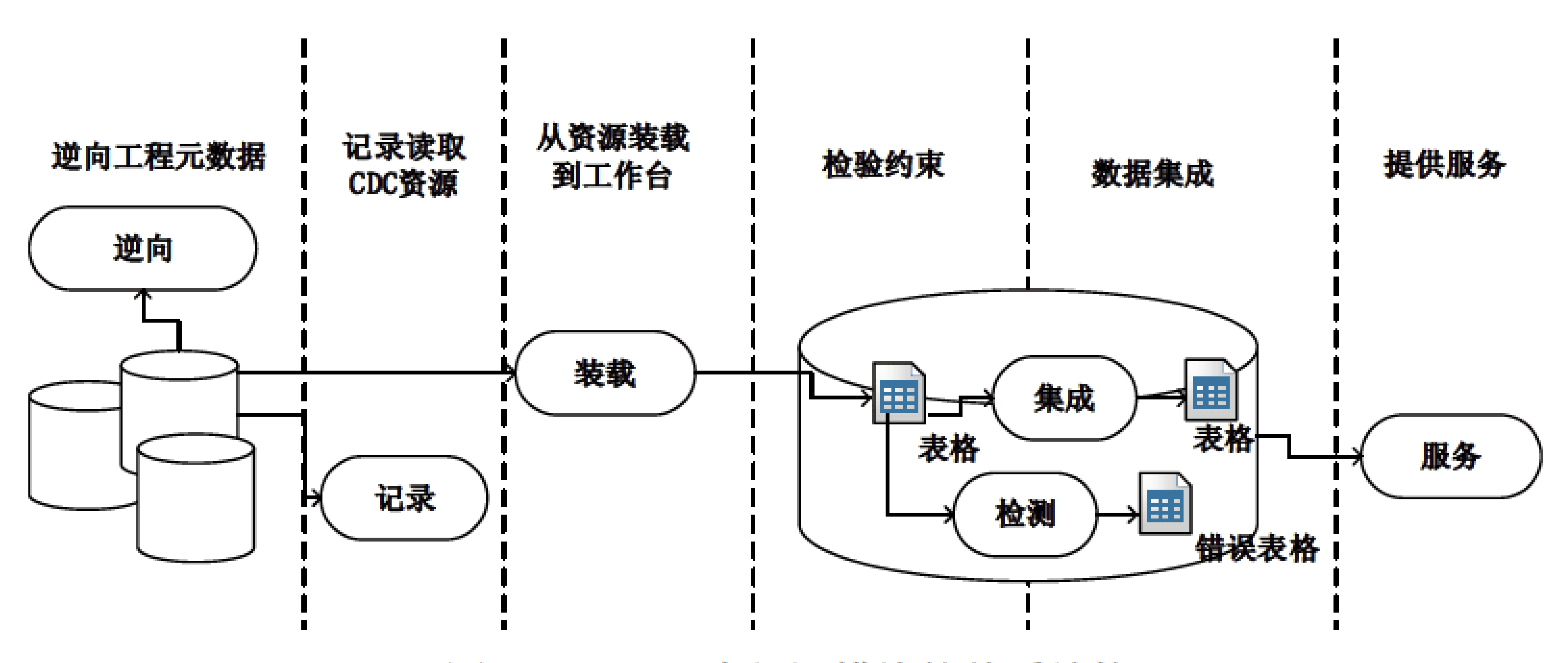


图 4.11　ODI 中知识模块的体系结构

- 反向工程（Reverse-engineering Knowledge Modules，RKM）：用于完成从数据源系统和目标系统的数据结构的反向工程来形成数据模型的功能。
- 日记（Journalizing Knowledge Modules，JKM）：用于为单一或一组表/视图记录新建的和修改的数据。
- 正在加载（Loading Knowledge Modules，LKM）：用于完成从源数据库中抽取数据，并加载到临时表的过程。
- 检查（Cheek Knowledge Modules，CKM）：用于完成对经过抽取得到的源数据的质量检查工作，检测其合法性。
- 集成（Integration Knowledge Modules，IKM）：完成将临时表中的数据转换加载到目标数据库中对应的数据表内。
- 服务（Service Knowledge Modules，SKM）：提供 ODI 与 WEB 服务的接口功能，并将数据以 Web Service 的方式展现出来。

4.3.2 ODI 的特点

与传统的 ETL 相比，ODI 在性能和成本等方面都存在着很大的优势。传统的 ETL 工具，其运行方式首先是从多种数据源中抽取数据，然后在一个专有的、中间层的 ETL 引擎转换数据，最后装载经过转换后的数据到数据仓库或集成服务器中。其中数据转换步骤是 ETL 过程中计算最为密集的步骤，执行时是由专有 ETL 引擎在专有服务器上完成的，导致在整个过程中容易造成瓶颈现象。而 ODI 不是采用独立的引擎，而是结合手工编码和 ETL 方法通过利用 RDBMS 的能力进行数据的转换，充分利用了数据库管理系统的能力和吞吐量，因此提供了最优的性能和可伸缩性，并且容易管理整个集成系统的架构。

ODI 体系中采用了声明式设计，即开发者只需要设计过程做什么，而不需要详细描述它怎样来做，可以有效地提高工作效率。

ODI 最大的优势是预置的、可热插入的知识模块。知识模块是给定集成任务的一个代码模块，这些代码是开放的，并且能够让技术专家通过图形化用户界面进行编辑来实现新的集成方法或最佳实践。

由此可见，利用 ODI 进行数据集成会给开发者带来很多的方便，优越性很高，但是其成本也是相当昂贵。另一方面，ODI 在具有超大数据量的情况下，进行批量加载时会有些显得力不从心。

4.3.3 Microsoft SQL Server Integration Services（SSIS）简介

SSIS 是 Microsoft SQL Server Integration Services 的简称，是生成高性能数据集成解决方案（包括数据仓库的提取、转换和加载包）的平台。

Integration Services 包括用于生成和调试包的图形工具和向导；用于执行工作流函数（如 FTP 操作）、执行 SQL 语句或发送电子邮件的任务；用于提取和加载数据的数据源和目标；用于清理、聚合、合并和复制数据的转换；用于管理 Integration Services 的服务；以及用于对 Integration Services

对象模型编程的应用程序编程接口（API）。SSIS 的体系结构如图 4.12 所示。

图 4.12　SSIS 系统结构图

SSIS 体系结构大体上由 5 部分组成，分别是：Integration Services 服务、Integration Services 对象模型、Integration Services 运行时和运行时可执行文件，以及封装的数据流引擎和数据流组件的数据流任务。其中数据流任务是 SSIS 中的一个核心任务。

- Integration Services 服务：用于监视正在运行的 Integration Services 包和管理包的存储。
- Integration Services 对象模型：用于访问 Integration Services 工具、命令行使用工具以及应用程序编程接口（API），用于创建在包中使用的自定义组件或用于创建、加载、运行和管理包的自定义应用程序。
- Integration Services 运行时：主要保存包布局，运行包，并为日志记录、断点、配置、连接和事务提供支持。
- Integration Services 运行时可执行文件：包括包、容器、任务以及 Integration Services 中包含的事件处理程序，同时还包括用户开发的自定义任务等。
- 数据流引擎和数据流组件：数据流任务封装了数据流引擎。数据流引擎提供将数据从源移动到目标的内存中的缓冲区，并调用从文件和关系数据库中提取数据的源，同时还管理修

改数据的转换以及加载数据或使数据可为其他进程所用的目标。Integration Services 数据流组件为 Integration Services 所包含的源、转换和目标组件，也可以在数据流中包含用户自定义的组件。

数据流任务包括三种不同类型的数据流组件，即源、转换和目标。其中：

- 源组件是一组数据存储体，主要包括关系数据库中的表、视图、文件（平面文件、Excel 文件、XML 文件等）；系统内存中的数据集；系统以外的外部数据等。
- 转换则是数据流任务的核心组件，包含了丰富的数据转换组件，如数据更新、聚合、合并、分发、排序、查找等。
- 目标组件与源组件相对应，也是一组数据存储体，用于存储目标数据。三种数据流组件之间则是通过“流（Flow）”组件相连，以完成数据流任务。数据流的执行过程可以认为是一个流水线过程，每一行数据都是装配线中需要处理的零件，而每一个转换都是装配线中的处理单元。图 4.13 能充分体现三者之间的关系。

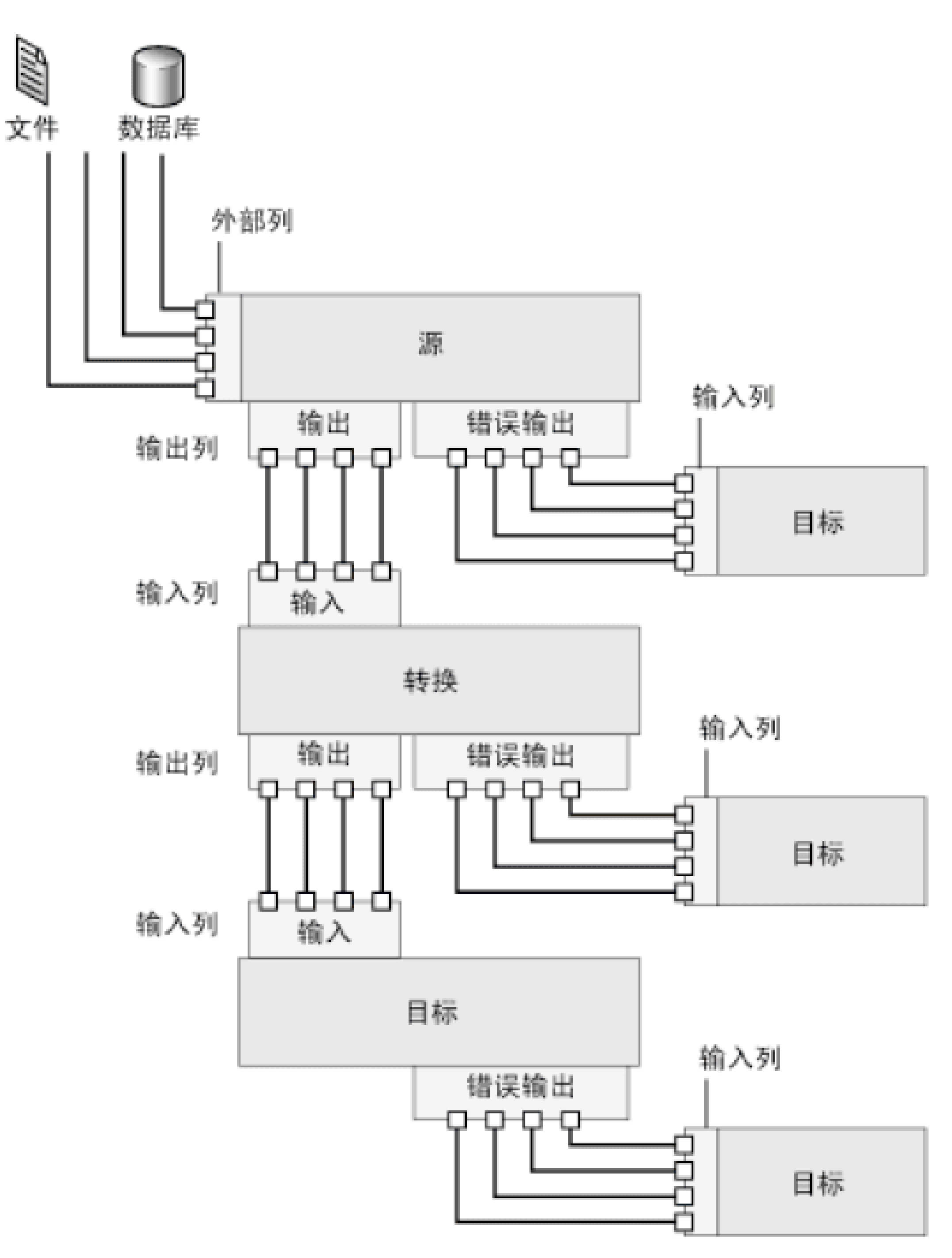


图 4.13　三种数据流组件之间的关系

在 SQL Server Integration Services 中使用图形设计工具或以编程生成方式将对象组合到包中。包是一个有组织的集合，包括连接、控制元素、数据流元素、事件处理程序、变量和配置等。用户可将完成的包保存到 SQL Server、SSIS 包存储区或文件系统中，用于将来被检索和执行。包是由一个控制流以及可选的一个或多个数据流组成的，图 4.14 所示为一个简单包，其中包含了一个带有数据流任务的控制流。

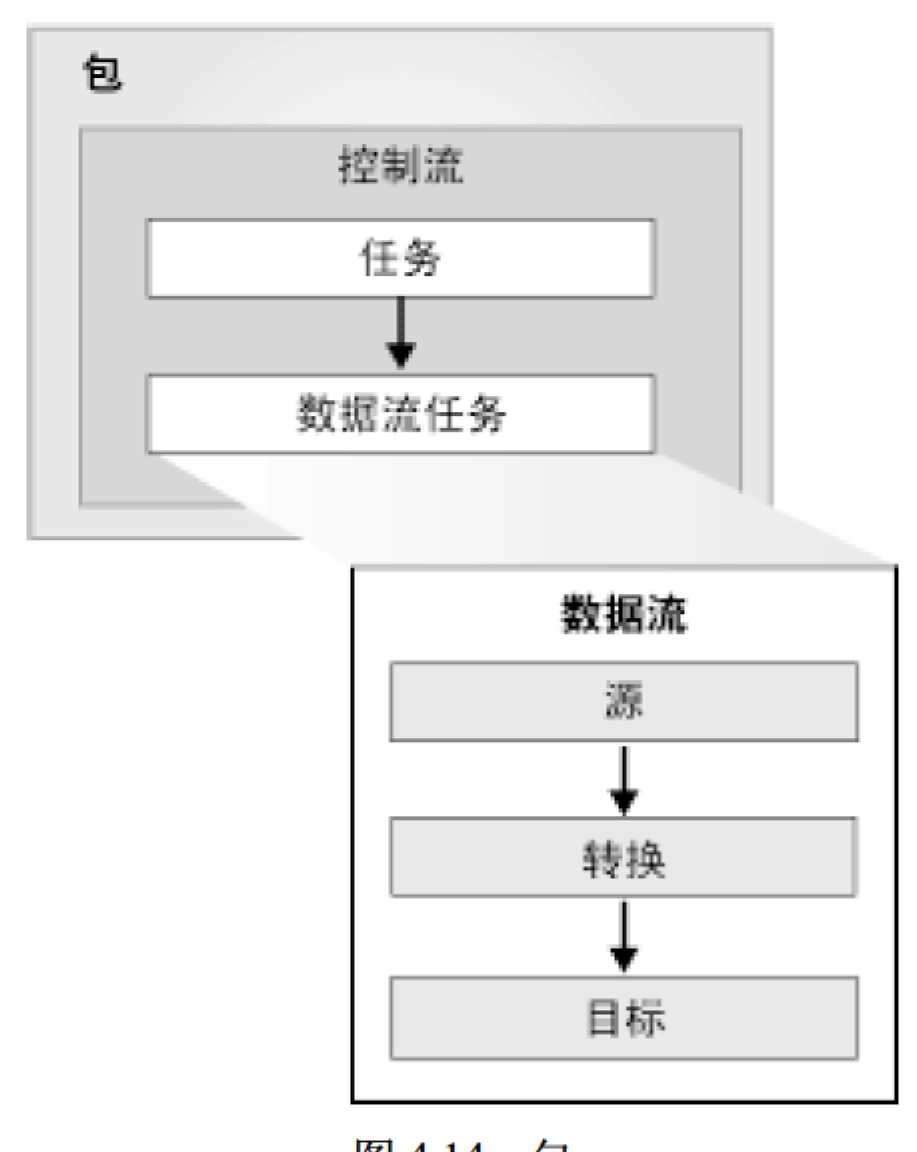


图 4.14 包

控制流包含了一个或多个在包运行时执行的任务和容器，主要负责高层的逻辑拓扑，完成对各个数据流单元的串接，因此控制流也可称为工作流或任务流。在工作流中每个组件都是一个任务，这些任务按预定义的数据线执行，能够标识出业务处理的先后顺序。任务流中存在多条分支，每条分支都决定这每个任务的执行结果。

SQL Server Integration Services（SSIS）中提供了三种不同类型的控制流元素：包中结构的容器、提供功能的任务以及将可执行文件、容器和任务连接为已排序控制流的优先约束。图 4.15 所示是具有一个容器和 6 项任务的控制流。

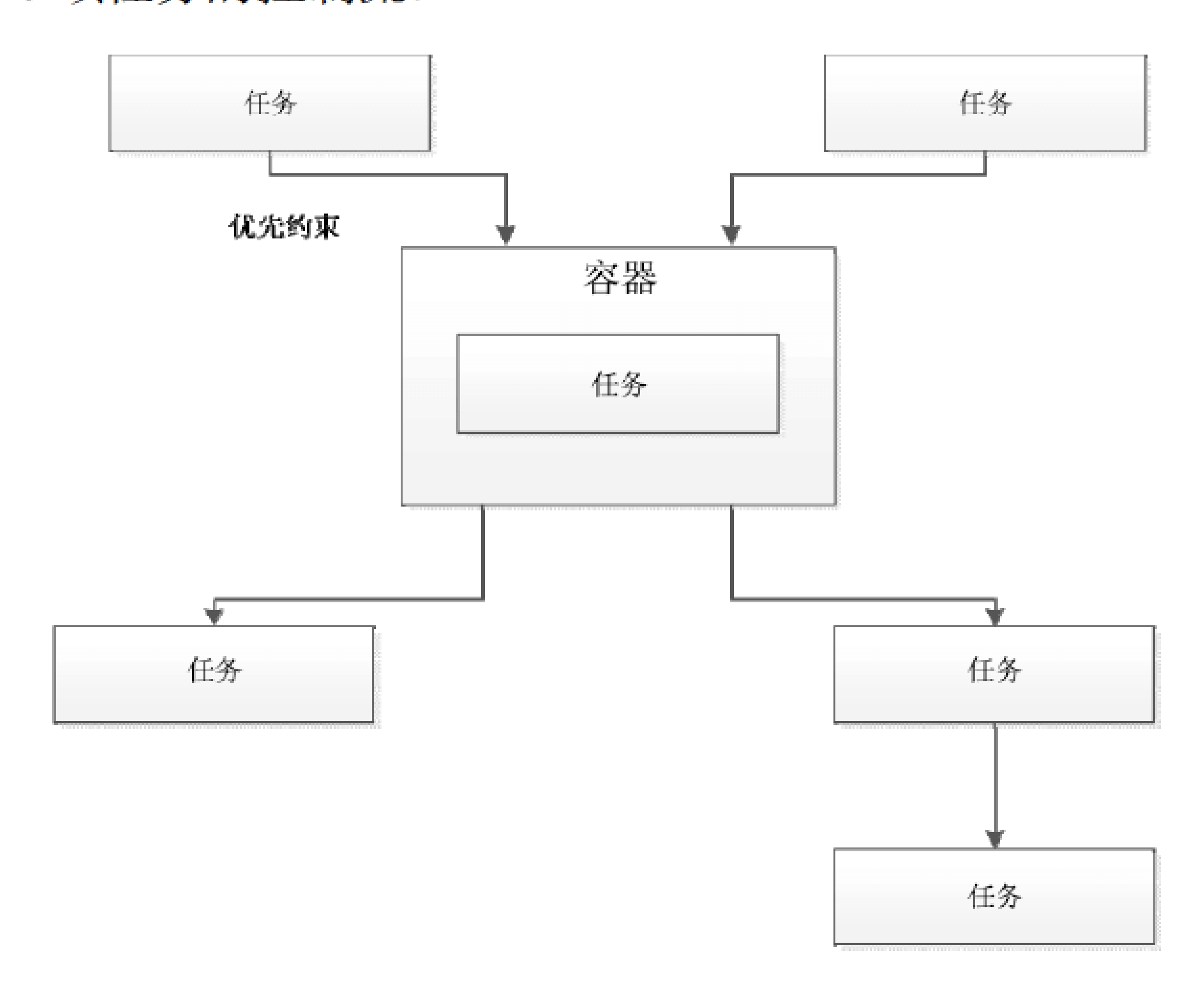


图 4.15 控制流

1. 容器

容器是为控制流中的任务提供包中结构和服务。Integration Services 中共包含三种类型的容

器，分别是 Foreach 循环容器、For 循环容器和序列容器，用于对任务分组以及实现重复的控制流。Foreach 循环容器是枚举一个集合，并对该集合的每个成员重复其控制流；For 循环容器作用是重复其控制流，直到指定表达式的计算结果为 False 为止；序列容器可以允许用户在容器内定义控制流子集，并将任务和容器作为一个单元来管理。

2. 任务

任务在包中用于执行工作。在 Integration Services 中包含执行多种功能的任务。下面简单介绍几类较为常用且重要的任务。

- 数据流任务：是定义并允许提取数据、应用转换和加载数据的数据流。
- 数据准备任务：用于复制文件和目录，下载文件和数据，保存由 Web 方法返回的数据或使用 XML 文档。主要包括文件系统任务、FTP 任务、Web 服务任务以及 XML 任务等。
- 工作流任务：与其他进程通信以运行包或程序，在包之间发送和接受消息，发送电子邮件等。具体有执行包任务、执行 DTS 2000 包任务、执行进程任务、执行消息队列任务、发送邮件任务等。
- SQL Server 任务：分别有大容量插入任务、执行 SQL 任务、传输数据库任务、传输错误消息任务、传输作业任务、传输登录名任务、传输主存储过程任务和传输 SQL Server 对象任务，主要用于访问、复制、插入、删除或修改 SQL Server 对象和数据。
- Analysis Services 任务：主要用于创建、修改、删除或处理 Analysis Services 对象。此类任务主要有 Analysis Services 处理任务、Analysis Services 执行 DDL 任务，以及数据挖掘查询任务。
- 脚本任务：允许用户自定义脚本，以扩展包的功能。
- 维护任务：主要用于执行管理功能，如备份和收缩 SQL Server 数据库、重新生成和重新组织索引以及运行 SQL Server 代理作业。具体包括备份数据库任务、检测数据库完整性任务、执行 SQL Server 代理作业任务、执行 T-SQL 语句任务、清除历史记录任务、通知操作员任务、重新生成索引任务、重新组织索引任务、收缩数据库任务及更新统计信息任务等。

另外，控制流中还允许用户使用支持 COM 的编程语言或.NET 编程语言编程自定义任务。

3. 优先约束

优先约束是将包中的可执行文件、容器和任务连接成为已排序的控制流，可以控制任务和容器的执行序列，并制定决定可执行文件、任务和容器是否运行的条件。可执行文件可以是 For 循环容器、Foreach 循环容器、序列容器、任务或事件处理程序。

4.3.4 SSIS 的特点

SSIS 超越 ETL 工具之处不仅在于它可以使用非传统的应用场景，而且在于它是一个可以实现数据集成真正的开发平台。SQL Server Integration Services、Analysis Services 和 Reporting Services

共同使用一个基于 Microsoft Visual Studio 的开发环境，即 SQL Server Business Intelligence （BI） Development Studio，其中 Integration Services 则是它们之间的中转站、枢纽点，能够将各种源头数据，经过 ETL 过程后导入到数据仓库，建立多维数据集，对其进行分析、挖掘并通过 Reporting Services 将结果展现给企业各级管理者。以下阐述关于 Integration Services 几个方面的特点。

1. 可视化环境

SSIS 的可视化环境允许通过简单的拖拽控件的可视化操作来实现有关 ETL 的大部分操作。这都得益于 SSIS 强大的控制流（Control Flow Function）以及灵活多样且高效的数据流任务（Data Flow Task）。正因为这些可拖拽的控件的存在，SSIS 也更加易于操作。SSIS 的可视化环境如图 4.16 所示。

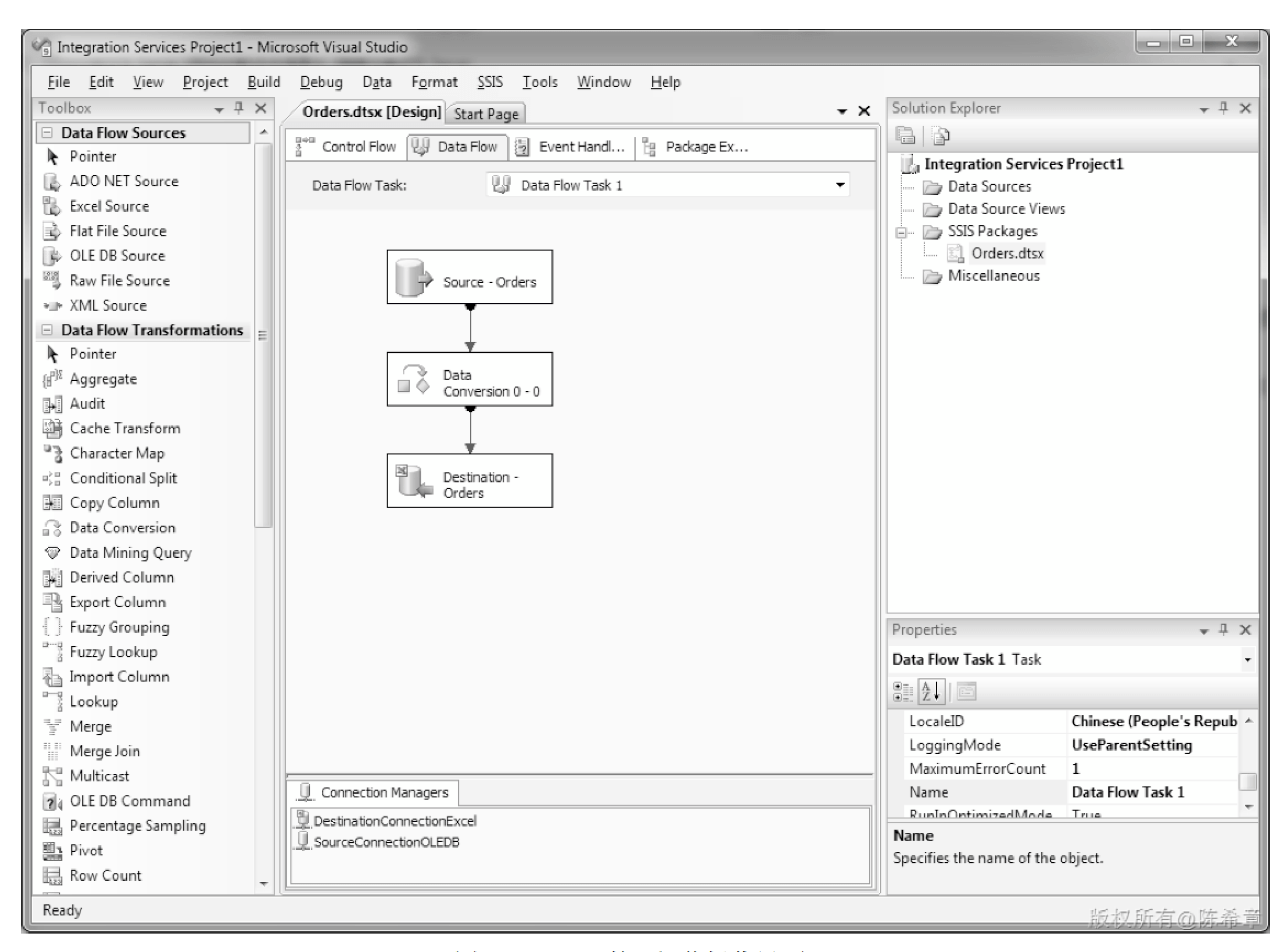


图 4.16　SSIS 的可视化操作界面

2. 具有参数设置功能

SSIS 的另一大特色之处是具有强大的参数设置功能，可以通过参数连接将数据源的表名与目标表的表名相关联，或通过参数来构建 SQL 语句中的条件子句。在一定程度上，能够简化 SSIS 基本的可执行单位 Package 程序的调用。

3. 可编程性

SSIS 除了提供的专业数据集成开发环境外，它还通过一套 API 展示其所有功能。API 既有托管的.NET Framework，也有本机的 Win32，能够让开发者使用任意一种.NET Framework 所支持的语言（Visual C#、Visual Basic、.NET 等）和 Visual C++支持的语言自定义开发组件用以扩展 SSIS 功能。SSIS 体现 SSIS 可编程性这一特点的同时，也突出地表现了它的可扩展性。

4. 可扩展性

SSIS 的可扩展特性是通过将自定义代码作为可以再度利用的扩展封装到 SSIS 中，并在此基础上充分利用日志记录、调试和 BI 集成等功能，将它们在自定义代码方面的现有投资用于数据集成。除此之外，SSIS 还可以获得基于脚本的可扩展性。SSIS 既有针对任务流的脚本组件，也有针对数据流的脚本组件，两者都允许用户用 Visual Basic 和.NET 语言编写脚本来添加其功能。

4.3.5 IBM InfoSphere Information Server 简介

IBM InfoSphere Information Server 提供数据集成的单一平台。套件中的组件组合在一起以创建企业信息体系结构的统一基础，能够进行扩展以满足任何信息量需求。可以使用该套件更快速地交付业务结果，同时在整个信息全景中维护数据质量和完整性。

IBM InfoSphere Information Server 有助于业务和 IT 人员进行协作，以了解各种源中信息的意义、结构和内容。 通过使用 InfoSphere Information Server，使用者可以采用新的方式访问和使用信息，以推动创新，提高运营效率并降低产品运营中的风险。

图 4.17 显示了 IBM InfoSphere Information Server 的关键功能，以供使用者用于实施完整的数据集成策略。这些功能的核心是一个公共元数据存储库，用于为 IBM InfoSphere Information Server 的所有组件存储已导入的元数据、项目配置、报告和结果。在元数据存储库中共享已导入的数据时，用户同一项目内的其他使用者可以使用其他 IBM InfoSphere Information Server 组件中的已导入资产并与其进行交互。

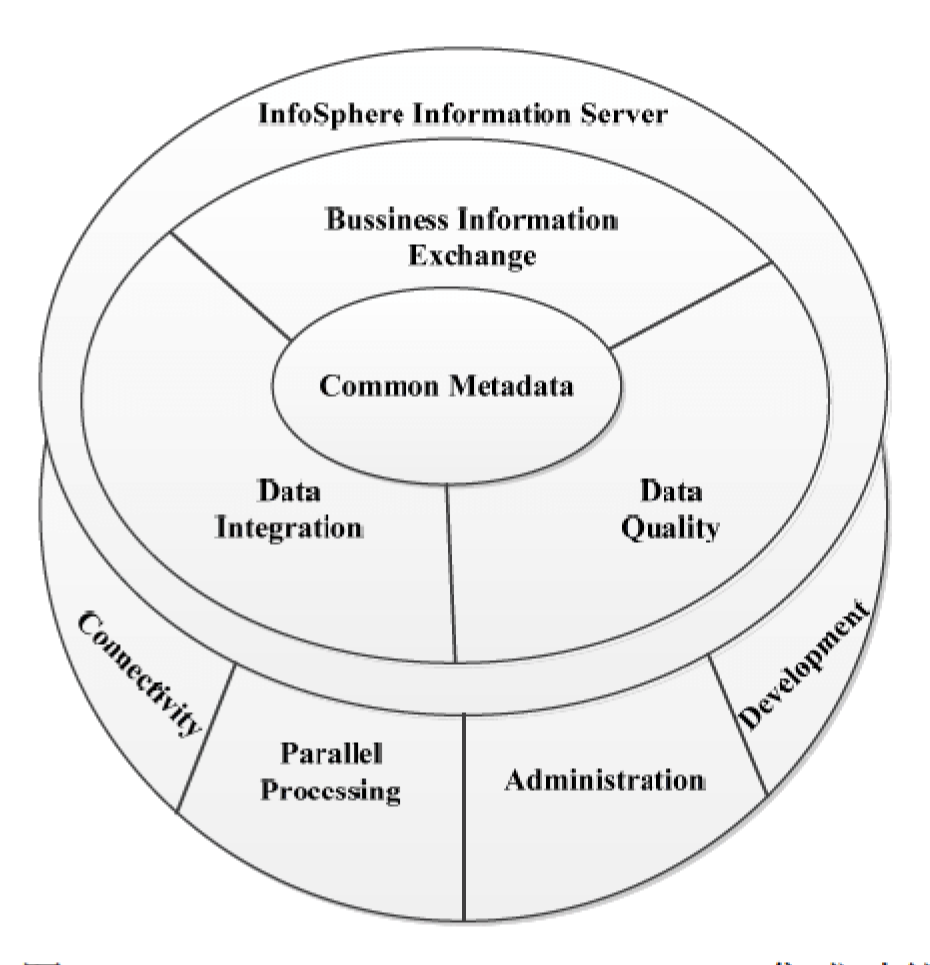


图 4.17 InfoSphere Information Server 集成功能

IBM InfoSphere Information Server 包括以下核心功能。

- 了解和协作

创建信息项目的蓝图，以开发业务的统一视图。使用该蓝图确定有助于保持业务透视图和 IT 透视图一致的公共业务语言。通过了解和分析信息的意义、关系和数据志来发现和定义现有数据源。

通过以数据志和质量证明支持完整、权威的信息视图，增强可视性和数据管理。可使这些视图作为共享服务广泛可用并且可复用，而其中固有的规则将进行集中维护。

- 清理和监视

以批处理方式实时标准化、清理和验证信息。将已清理的信息装入到分析视图，以监视和维护数据质量。在整个企业复用这些视图，以确定符合业务目标的数据质量度量，从而支持组织快速发现和修正数据质量问题。

在系统之间链接相关的记录，以确保信息的一致性和质量。将不同数据整合到单个可靠的记录中，以确保最佳数据在多个源中继续存在。将此主记录装入到运作数据存储、数据仓库或主数据应用程序中以创建可信的信息源。

- 变换和交付

设计和开发数据集成项目的蓝图，以提高可视性并降低风险。发现系统之间的关系，并定义在多个源和目标之间集成资产元数据的迁移规则。了解关系并集成数据可降低运营成本并提升数据质量。

收集、变换和分发大量数据。使用内置变换功能来缩短开发时间，提高可伸缩性并提供设计灵活性。通过批量数据交付（ETL）、虚拟数据交付（联合）或增量数据交付（更改数据捕获）向业务应用程序实时交付数据。

1. 信息集成阶段

IBM InfoSphere Information Server 专注于作为有效信息集成项目组成部分的几个阶段。这些阶段随项目生命周期的发展和更改而不断演进。通过提供关键信息集成功能，IBM InfoSphere Information Server 能够处理其中每个阶段以确保项目成功。

图 4.18 显示套件组件如何一起工作以创建统一的数据集成解决方案。公共元数据基础使不同类型的用户能够通过使用针对其角色优化的工具来创建和管理元数据。通过这种对个性化工具的专注，更易于在角色之间进行协作。

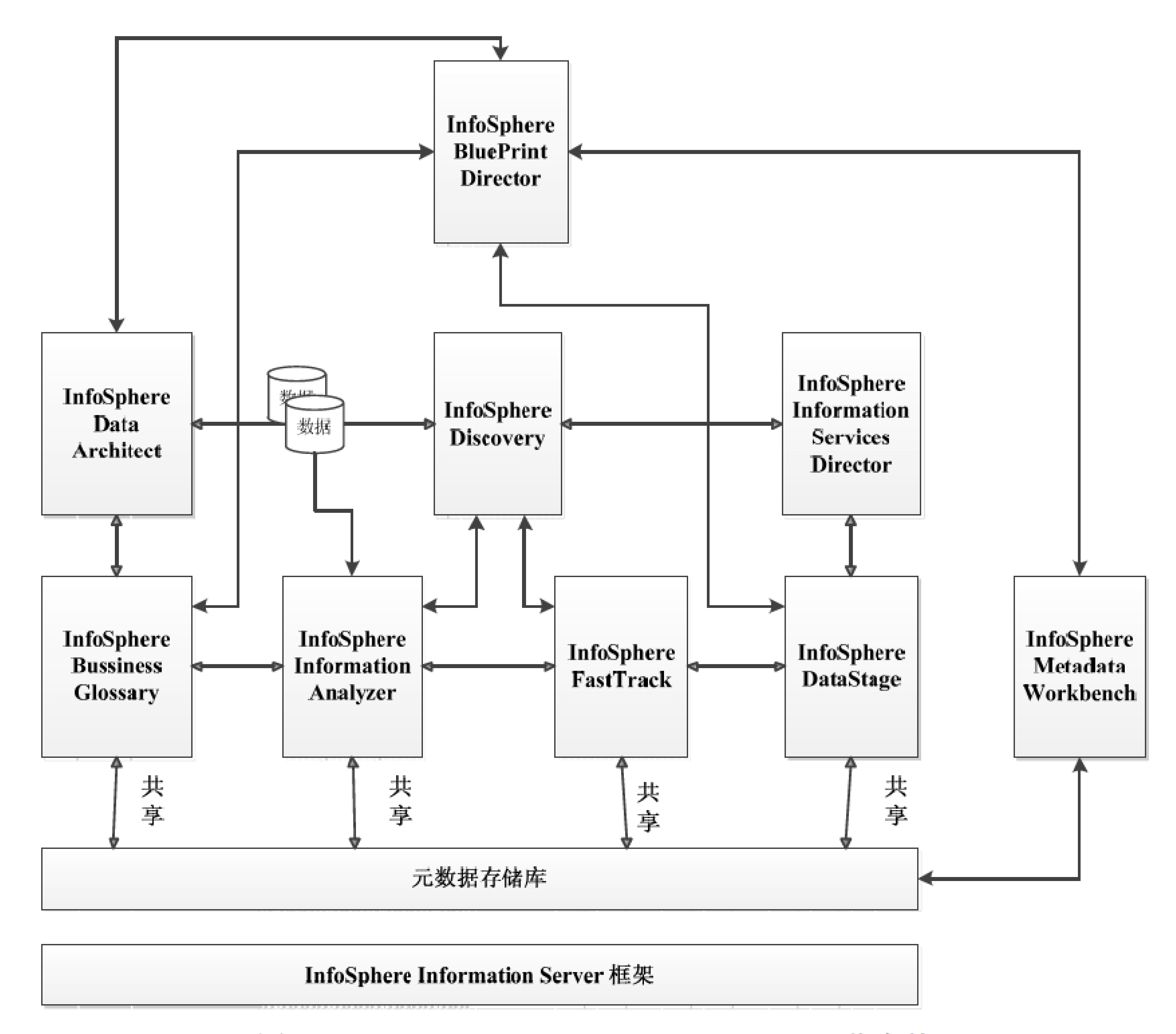


图 4.18 IBM InfoSphere Information Server 工作套件

企业架构设计师使用 InfoSphere Blueprint Director 来规划和管理项目远景。存在信息项目的蓝图后，数据架构设计师可以使用 InfoSphere Data Architect 来发现组织的数据的结构、关联和集成数据资产以及基于这些关系来创建物理和逻辑模型。 该数据可以输入到 InfoSphere Business Glossary 中，其中业务分析员和数据分析员定义和确定业务概念的共同理解。

数据分析员还可以使用 InfoSphere Discovery 来自动执行数据关系的确定和定义，以为 InfoSphere Information Analyzer 和 InfoSphere FastTrack 供应该信息。

数据质量专家使用 InfoSphere Information Analyzer 来设计、开发和管理需要管理的数据的质量规则，以确保数据质量。随着组织数据的不断发展，可以对这些规则进行实时修改，以便将可信的信息交付给 InfoSphere Business Glossary、InfoSphere FastTrack、InfoSphere DataStage and QualityStage 和其他 InfoSphere Information Server 组件。

数据分析员可以使用 InfoSphere FastTrack 来创建将业务需求转换为业务应用程序的映射规范。数据集成专家可以使用这些规范来生成将成为 InfoSphere DataStage and QualityStage 中复杂数据变换的起始点的作业。通过使用 InfoSphere DataStage and QualityStage Designer，数据集成专家可以开发用于抽取、变换、装入数据并检查数据质量的作业。 SOA 架构设计师使用 InfoSphere Information Services Director 将套件组件中的集成任务部署为一致且可复用的信息服务。

IBM InfoSphere Metadata Workbench 提供使用者的数据资产的端到端数据流报告和影响分析。业务分析员、数据分析员、数据集成专家和其他用户与该组件进行交互，以浏览和管理 IBM InfoSphere Information Server 生成和使用的资产。

InfoSphere Metadata Workbench 支持用户了解和管理整个企业中的数据流，并发现和分析 InfoSphere Information Server 元数据存储库中信息资产之间的关系。 用户可使用 InfoSphere Metadata Asset Manager 将技术信息导入到元数据存储库中，如 BI 报告、逻辑模型、物理模式以及 InfoSphere DataStage and QualityStage 作业。

2. 套件中的组件

IBM InfoSphere Information Server 套件包含许多组件，每个组件都提供不同的数据集成功能。 连接在一起后，这些组件将形成在整个企业内交付可信信息所必需的构建块，而与环境的复杂性无关。

IBM InfoSphere Information Server 解决方案包含了不同的组件以满足使用者的需求。 每个解决方案都包含 InfoSphere Blueprint Director、InfoSphere Discovery 和 InfoSphere Metadata Workbench 作为基础组件。每个解决方案中的其他组件提供专注于数据质量、数据集成以及连接业务用户和 IT 用户的不同功能，如表 4.1 所示。

表 4.1　InfoSphere Information Server 解决方案中包含的组件

组件	InfoSphere Information Server 商业信息交流	InfoSphere Information Server 数据集成平台	InfoSphere Information Server 数据质量平台
InfoSphere Blueprint Director	✓	✓	✓
InfoSphere Discovery	✓	✓	✓
InfoSphere Metadata Workbench	✓	✓	✓
InfoSphere Data Architect	✓	✓	
InfoSphere Business Glossary	✓		
InfoSphere Business Glossary Anywhere	✓		
InfoSphere Information Analyzer		✓	✓
InfoSphere QualityStage®		✓	✓
InfoSphere Information Services Director		✓	✓
InfoSphere DataStage and QualityStage Designer		✓	✓
InfoSphere Data Click		✓	
InfoSphere FastTrack		✓	
InfoSphere Data Replication		✓	

3. InfoSphere Information Server 产品服务组合中的其他组件

在定义数据集成策略中主动发挥作用是实现业务目标的关键。不论用户的目标是数据质量、数据集成、连接业务与 IT 还是上述项的某种组合，IBM InfoSphere Information Server 都有很好的扩展性，以满足用户的需求。

4. IBM InfoSphere Information Server 体系结构和概念

IBM InfoSphere Information Server 提供可处理所有类型的信息集成的统一体系结构。公共服务、统一并行处理以及统一元数据是服务器体系结构的核心。

体系结构是面向服务的，使 IBM InfoSphere Information Server 能够在不断发展的面向企业服务的体系结构中发挥作用。面向服务的体系结构也连接 InfoSphere Information Server 的各个套件产品模块。

通过消除重复的功能，此体系结构有效地使用硬件资源，并减少部署集成解决方案所必需的开发和管理的工作量。

图 4.19 显示了 InfoSphere Information Server 体系结构。

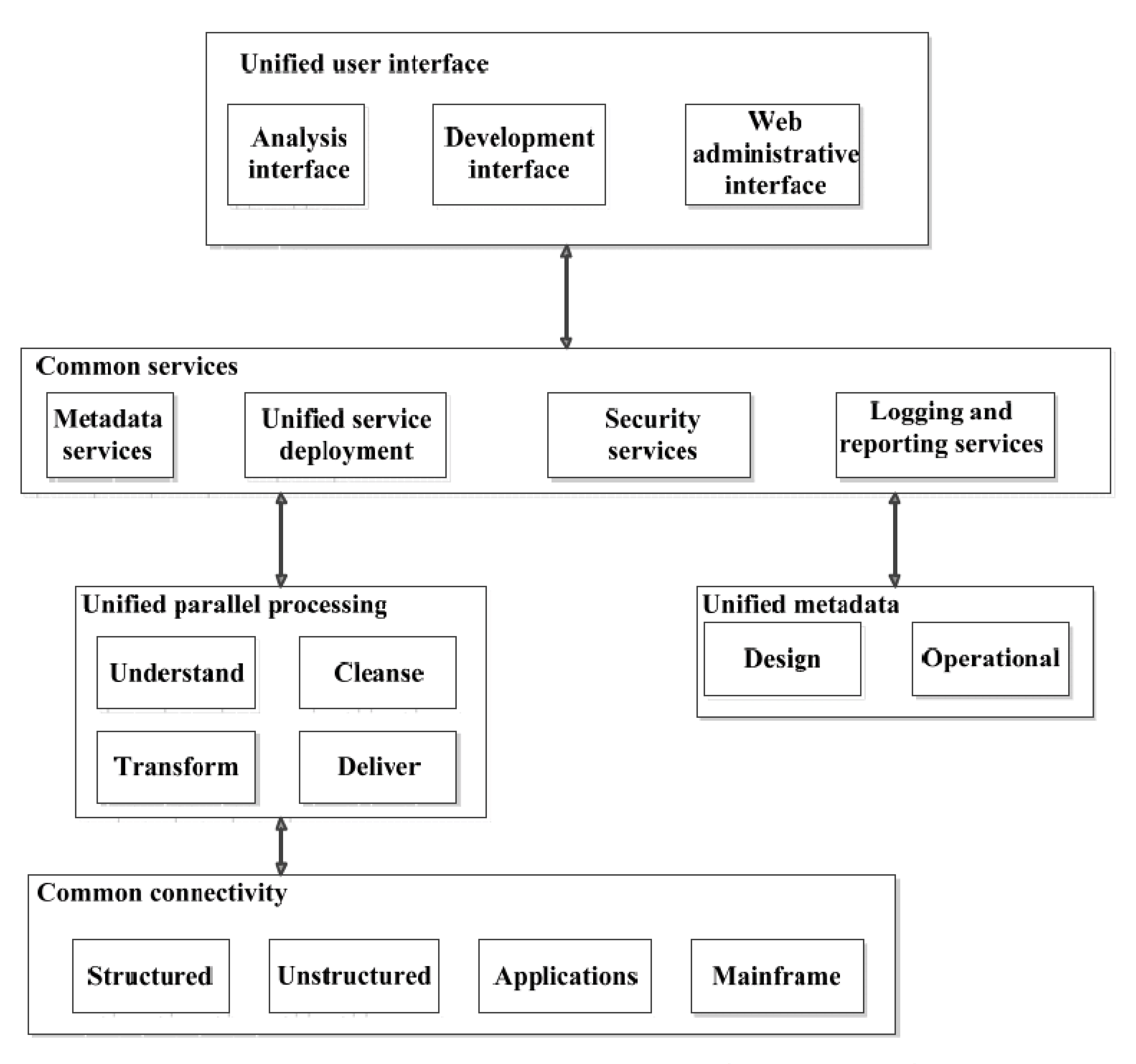


图 4.19 InfoSphere Information Server 高级别体系结构

（1）统一并行处理引擎

InfoSphere Information Server 的大多数工作都在并行处理引擎中进行。引擎不仅处理数据处理需求，而且还对 IBM InfoSphere Information Analyzer 执行大型数据库分析，对 IBM InfoSphere QualityStage 执行数据清理，并对 IBM InfoSphere DataStage 执行复杂变换。此并行处理引擎的

设计目的是提供以下优势：

- 并行性和数据流水线，以便在不断缩短的时间期限内完成不断增加的大工作量。
- 可伸缩性，通过添加硬件（例如，网格中的处理器或节点）而不更改数据集成设计而实现。
- 经优化的数据库、文件以及队列处理，用于处理不能一次全部放入内存的大型文件或大量的小型文件。
- 公共连接，无论信息源是结构化的还是非结构化的，在大型机上还是应用程序上，InfoSphere Information Server 都可以连接到这些信息源。元数据驱动的连接在套件组件之间共享，而且连接对象可在各功能之间复用。

连接器提供设计时元数据导入、数据浏览和采样、运行时动态元数据访问、错误处理以及高功能和高性能运行时数据访问。名为 packs 的封装应用程序的预构建接口为 SAP、Siebel、Oracle 以及其他产品提供了适配器，实现了与企业应用程序及关联报告和分析系统的集成。

（2）统一元数据

InfoSphere Information Server 在统一元数据基础结构上进行构建，该基础结构使业务领域和技术领域之间能够共享一些协定。该基础结构减少开发时间并提供可提高信息置信度的持久记录。InfoSphere Information Server 的所有功能共享同一个元模型，这使得不同角色和功能间的协作更加容易。

公共元数据存储库为所有 InfoSphere Information Server 套件组件提供了持久存储。所有产品都依赖于存储库来浏览、查询和更新元数据。该存储库包含两种元数据：

- 动态元数据。动态元数据包含设计时信息。
- 操作元数据。操作元数据包括性能监视、审计和日志数据以及数据概要分析样本数据。

例如，因为存储库供所有套件组件共享，所以 InfoSphere Information Analyzer 创建的概要分析信息将对 InfoSphere DataStage 和 InfoSphere QualityStage 的用户即时可用。

存储库是一个 J2EE 应用程序，它使用 IBM DB2、Oracle 或 SQL Server 之类的标准关系数据库进行持久保存（DB2 随 InfoSphere Information Server 一起提供）。这些数据库提供备份、管理、可伸缩性、并行访问、事务以及并发访问。

（3）公共服务

InfoSphere Information Server 完全是在一组共享服务上构建的，这些服务集中了平台上的核心服务。它们包括各种管理任务，例如安全性、用户管理、日志记录以及报告。共享服务允许在一个位置对这些任务进行管理和控制，而不考虑正在使用的套件。公共服务还包括元数据服务，该服务提供了平台上标准的面向服务的访问和元数据分析。此外，公共服务层还管理如何从产品功能中部署服务，通过使用一致和易于使用的机制，使得清理和变换规则或联合查询能够在 SOA 中发布为共享服务。

InfoSphere Information Server 产品可以访问三个常规类别的服务：

- 设计。设计服务可帮助开发者创建同样能够共享的有特定功能的服务。例如，InfoSphere

Information Analyzer 调用列分析器服务，该服务是为企业数据分析而创建的，但可与 InfoSphere Information Server 的其他部分集成，因为它呈现了公共 SOA 特征。

- 执行。执行服务包括日志记录、调度、监视、报告、安全性和 Web 框架。
- 元数据。元数据服务支持在各个工具之间共享元数据，因此，在一个 InfoSphere Information Server 组件中所作的更改会在所有套件组件中即时可见。元数据服务与元数据存储库集成。元数据服务还支持与外部工具交换元数据。

公共服务层部署在符合 J2EE 的应用程序服务器上，例如 InfoSphere Information Server 附带的 IBM WebSphere Application Server。

（4）统一用户界面

InfoSphere Information Server 的风格是公共图形界面和工具框架。共享界面（例如 IBM InfoSphere Information Server 控制台和 IBM InfoSphere Information Server Web 控制台）在不同产品之间提供了公共的界面、可视控件和用户体验。常见功能（例如目录浏览、元数据导入、查询以及数据浏览）都以统一方式显示了底层的公共服务。InfoSphere Information Server 提供了富客户机接口以执行非常详细的开发工作，并提供了在 Web 浏览器中运行以进行管理的瘦客户机。

应用程序编程接口（API）支持各种接口样式，包括标准请求/应答、面向服务、事件驱动以及计划任务调用。

4.3.6 Sybase Data Integrator Suite 简介

如今，企业迫切希望 DBA（数据库管理员）和开发人员能够集成公司数据，以便协助管理信息、挖掘客户数据库或满足日常要求。Sybase 正借助一种称为 Sybase 数据集成 （DI） 套件的新产品来满足这种需求。此项新技术的主要功能包括：

- 访问多个不同数据源，且能够创建单一、集成的数据视图。
- 访问各种异构数据源，包括大型机数据源。
- 捕获数据源中的实时事件，并将其传播到应用程序中。
- 使用上下文搜索对结构化和非结构化数据中的信息进行搜索和查询。
- 使用 Sybase WorkSpace 开发应用程序。
- 使用通用的系统管理控制台管理 DI 套件组件。
- 通用的安装程序，它使用脚本驱动的实用程序来执行交互式和后台安装。

Sybase DI 套件包含集成数据的所有常用技术（联邦、复制和 ETL）以及实时和搜索功能。图 4.20 说明了数据集成套件的各种组件。

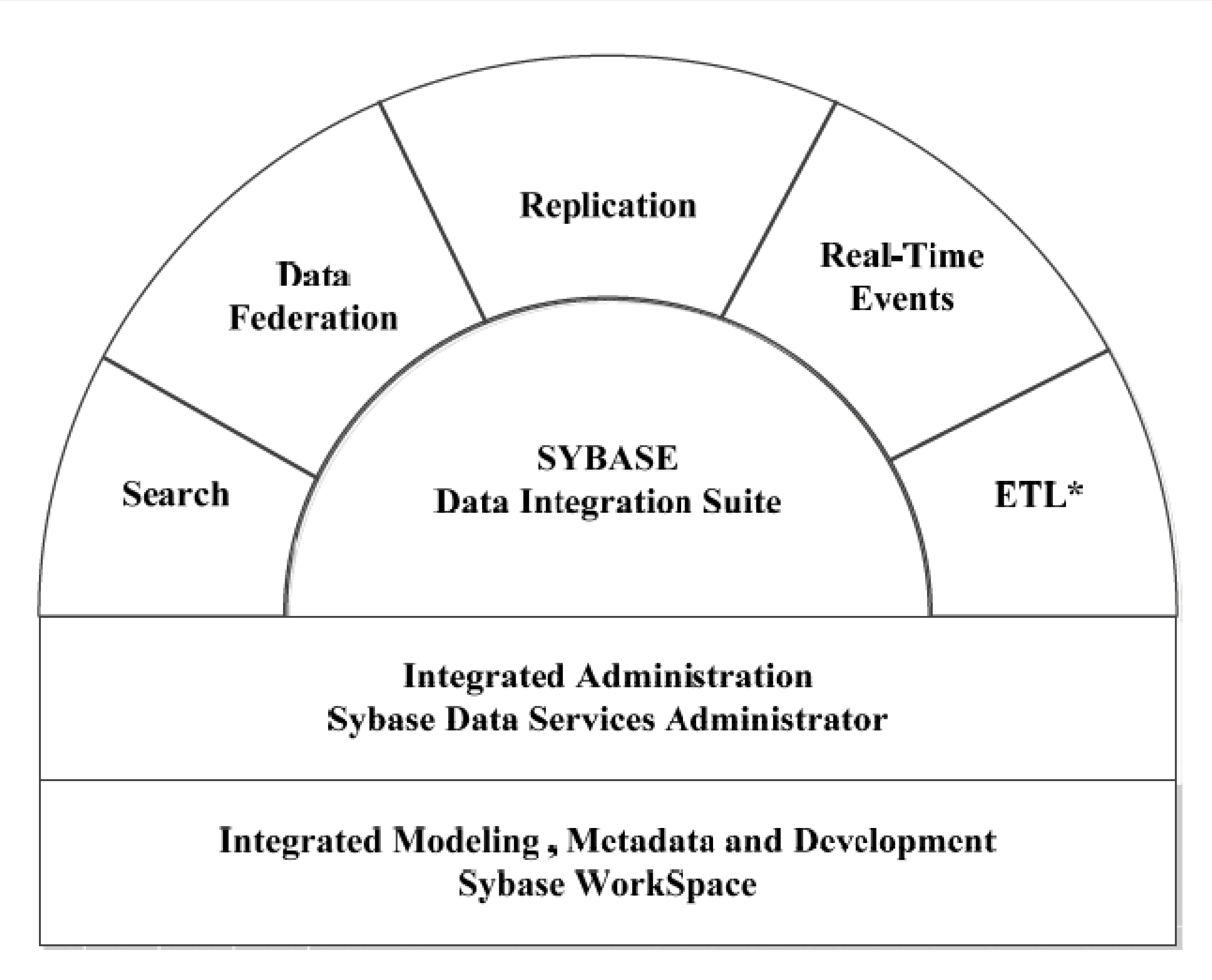


图 4.20 Sybase DI 套件

1. 企业信息集成

由 Sybase 最近兼并的 Avaki 公司提供的 Sybase 数据联邦是 DI 套件的一个关键组件。它基于日益成为主流的企业信息集成（EII）概念，企业信息集成与以 ETL 为导向的数据仓库不同，因为它只访问数据而不移动信息。ETL 本身就是一个负责将数据移动到支持商业智能报告的中心存储库或数据集市，由很多步骤组成的过程。

然而，尽管 EII 使用虚拟化来显示统一的信息资源，但事实上它是调用来自多个资源的“联邦”数据而不是进行拷贝。随着数据库数量不断增加，联邦数据方式在访问分布式数据方面变得越来越重要。Avaki EII 为用户提供了 EII 技术的许多核心功能，并增加了网格功能、数据高速缓存、独立地跨防火墙和管理域移动数据的功能以及共享文件的功能。

2. 数据联邦

如上所述，联邦简化了集成来自多个分布式来源的数据的过程，且能够访问集成的企业数据。用户可以通过它：

- 获取来自多个不同来源的数据的单一虚拟视图，这些来源包括支持 JDBC/ODBC 访问的关系数据库、应用程序（通过 JCA 或者 Web 服务）、Web 服务、XML 文档或文件。“联邦”方法意味着，数据是从原始数据源中提取出来的，而不是从数据副本或数据集市中提取出来的。
- 将联邦数据以 Web 服务、SQL 视图（使用 JDBC/ODBC 访问）或者平面文件的形式提供给需要使用这些数据的应用程序。因此，现在应用程序无需创建联邦查询或者直接访问源数据库，但可以访问数据联邦服务器以执行其查询。
- 利用基于 GUI 的工具定义数据源和查询，并在目录中存储这些查询以便于由应用程序执行或者搜索和重用。它还支持即时查询。
- 更改数据源的模式或者将数据库移动到其他服务器中，且不破坏应用程序。只需对应用程序正在使用的 Web 服务或者 SQL 视图进行修改即可。

- 依靠用户身份验证的安全支持以及在EII层中定义的与查询相关的常用企业验证模式和精细的访问控制。
- 确定哪些用户登录了服务器及他们运行了什么样的查询。
- 为面向服务的应用程序构建数据服务层，使得数据层和业务逻辑分离。

DI 套件还包含了提供 EII 功能和工具的数据联邦服务器。用户分别使用套件安装程序来安装 EII 功能，使用 Sybase WorkSpace 来安装工具。

3. 复制

复制组件是 DI 套件的数据分发和数据同步组件，它包含了 Sybase 复制服务器的所有组件。它具有以下功能：

- 支持异构数据库（包括 ASE、Oracle、IBM DB2 和 Microsoft SQL Server）中事务数据的移动和同步。
- 提供读取上述任意数据库的功能，并将变化传播到相同的或不同的目标数据库中。
- 性能极高，由于采用从日志文件读取数据库变化，而非基于触发器的方法时，所以不会影响数据库的正常运行。
- 以事务处理的方式实时地将事务从源数据库传播到目标数据库(即不向目标数据库提交在源数据库上回滚的事务)。此外，复制在事务级别上维护数据完整性，从而确保只向目标数据库提交完整事务。
- 在分布式异构系统间启用双向复制。
- 根据需要允许转换正在使用的数据。
- 准许将源数据库中的模式变化传播到目标数据库中。
- 复制是非常灵活的，它允许客户指定是复制整个数据库、整个表格还是仅复制表中指定的列。

4. 实时事件

实时事件通过消息基础架构从异构数据库中捕获限时事件，并将其推入业务应用程序中。有了它就不再需要基于轮询的应用程序了，因为这些应用程序会影响生产服务器的效率。

此组件具有以下功能：

- 能够从各种数据库(Sybase ASE、Oracle)中捕获事件，并将这些变化传播到消息总线中。还可以读取消息总线中的事件，并将其作为 SQL 语句在数据库中应用。
- 允许应用程序捕获和添加与事务的状态相关的信息(即与事务相关的其他信息)，并将其作为 XML 消息推入消息总线中。
- 支持标准的消息基础架构，例如 Java 消息服务（JMS）以及传入和传出 WebSphere MQ 的消息服务。

5. 搜索

搜索组件提供高级数据服务以便查询、定位和分析数据。它自动处理、定位并分析数据库、集中式存储库、程序库、文件系统、网络驱动器和现有文档管理系统中相关性最大的信息。

用户可以通过它：

- 自动捕获和聚集非结构化数据；检索并提供相关内容。其数据导入功能包括文件系统、Web 数据和数据库。
- 支持多种格式，包括 Microsoft Word、Excel 和 PowerPoint、纯文本文件、Adobe Acrobat PDF 文件以及 HTML 文件。
- 以自然语言进行搜索。
- 进行智能处理。它可以推断文档中的概念。软件可以搜索文档、段落和元数据，并对它们进行分类。
- 个性化。搜索根据设定的兴趣或用户配置文件自动提供内容。
- 与语言无关。可以用各种语言进行搜索，且不是基于关键字的。

6. ETL（提取、转换、装载）

目前，ETL 是独立于数据集成套件销售的。需要传输数据的用户可以单独购买。用户可以使用它访问异构数据源（数据库、XML 文件等）、转换数据并将其装载到各种目标数据库（数据仓库、XML 文件等）中。

- 目前所支持的数据源和目标数据库包括 Oracle、DB2、MS SQL Server、Sybase ASE、Sybase IQ、XML 和文本文件。可以使用 Sybase 适配器提取大型机数据，并将其写入 ETL 过程将使用的文件中。ETL 工具支持各种服务，包括 SOAP 和 XMLRPC。
- 图形化开发界面使用户可以利用拖放功能创建转换流。用户可以使用不同的组件进行输入、转换、查找、分级和输出。对于每个组件，都提供 Flash 教程和每一执行流程的各种向导。此外，仿真环境提供完成 ETL 工作的步骤，并检查每个组件的输入、输出和转换。为 JavaScript 开发提供全面的调试工具，此工具允许逐行分步执行、定义监测点、执行白盒测试和评估表达式。
- ETL 工具可以读取平面文件和通过特定更改完成频繁的批处理更新。
- 转换组件包含分步控制、映射、拆分和转换数据流、JavaScript 以及调用外部代码的功能。
- 作业控制组件允许用户管理相关性和同步转换过程。

7. Sybase 数据管理服务器

Sybase 数据服务管理器（DSA）是管理数据集成（DI）套件组件的图形界面。它提供以控制面板形式组织的可视 DI 套件组件，包括可通过 Sybase Central 插件访问的、基于 Sybase Central 界面的服务器管理器。Sybase Central 界面如图 4.21 所示。

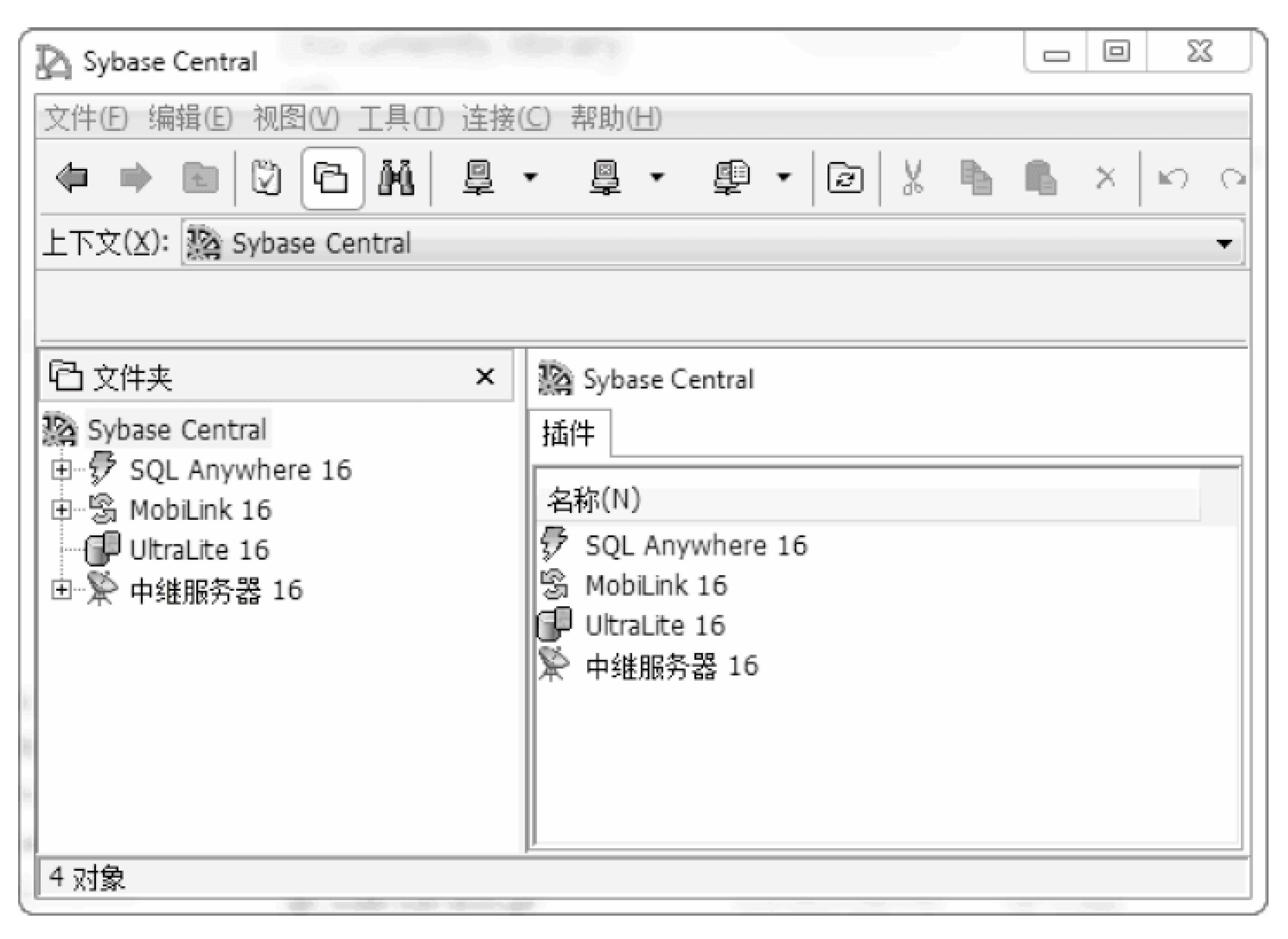


图 4.21 Sybase Central 界面

8. Sybase WorkSpace

Sybase WorkSpace 是 DSA 提供显示有关执行特定管理任务的信息的在线帮助。安装任何 DI 套件组件之后，即可访问此帮助。

WorkSpace 提供开发 DI 套件的数据联邦、复制和实时事件组件的功能。这只在 Windows 操作系统中可用。

WorkSpace 可以作为 DI 套件中的联邦（Avaki EII）服务器的工具。其用于数据集成的元数据驱动的图形建模工具将帮助用户集成来自异构数据源的数据。可以构建数据服务，提供或搜索数据源，以及导入或创建数据服务的模式模型。还可以使用 WorkSpace 企业建模工具中的 DataArchitect 进行反向工程或创建新模式模型，然后将它们导入数据联邦工具中，以便创建符合这些模型需要的数据服务。

用户还可以通过 WorkSpace 管理和建立异构复制系统。用户创建适用于复制系统的信息流动模型，它将自动管理复制定义、发布和订阅（创建完成之后，用户可以手动修改这些设置）。用户还可以通过企业建模工具反向工程现有复杂的复制环境，以执行影响分析。

对实时事件管理而言，WorkSpace 的工具有助于捕获实时的数据库事件。

9. Sybase 数据集成套件解决方案

数据集成套件组件提供了解决企业中不同数据集成难题的工具。

表 4.2 提供了数据集成问题的列表，包括问题的详细信息以及 DI 套件提供的解决方案。

表 4.2　常见数据集成问题及解决方案

问题	详细信息	解决方案
数据孤岛	有价值的数据陷入不同的数据源或不兼容的数据模式中。现有系统导致这些数据不可访问。 不同数据源中的数据没有被集成起来。 由于数据源之间的事务变化未被分发和同步，所以无法访问实时数据	复制连接所有支持的异构数据存储，保持数据的集成性和近乎实时的访问数据
没有统一的企业数据视图	访问生产数据将影响系统的运行性能。 数据仓库中的数据并不是实时的。 没有来自不同数据源的生产数据的统一视图，这将影响那些对访问时间敏感的应用程序获取关键数据	数据联邦有助于： ● 获得各种数据的单一视图 ● 实时查看生产数据 ● 创建不同的视图以访问操作数据和数据仓库数据
无法自动使用实时数据和历史数据以提高业务流程的效率和质量	商务智能（BI）工具只访问数据仓库，不访问生产数据	数据联邦将帮助联邦数据仓库和生产系统中的数据，以获得当前数据和历史数据的综合视图
无法识别并防止对服务的未授权使用	开发人员编写服务并在其代码中嵌入访问逻辑。 无法在代码中添加权限	数据联邦能够创建用户并设置每种服务的访问权限。 这将使得嵌入在代码中的业务逻辑与服务访问权限分离
没有能够处理企业中大量非结构化信息的单一解决方案	根据文章的内容手工做标记以便搜索。这项工作的劳动强度非常高，且需要大量的标记。 在搜索相似内容时，使用搜索引擎执行关键字搜索而非基于内容的搜索。关键字搜索将显示在不同文档内容中所有与关键字匹配的结果	DBA 可以通过搜索组件来搜索非结构化数据和关系数据库中的数据。此组件能够自动根据文档的内容对文档进行分类，并以自然语言搜索包含相似文本的文档
企业兼并之后没有该企业的历史数据	每个被收购公司生成它们自己的报告。 整合这些报告不仅困难且非常耗时	DI 套件提供以下选择： ● 将所有数据复制到中心数据库，并创建集合视图。 ● 联邦访问不同企业中的数据
没有实时同步信息，这将对使用应用程序访问数据的用户产生影响	应用程序无法获得数据源中发生的事件。 只按一定时间间隔将数据变化更新到中心存储库中。因此，数据并不是最新的数据	实时事件组件可以捕获源数据库中的变化，并将其作为事件推入消息总线中。订阅这些事件的应用程序将把此变化应用到目标数据库中

复制、联邦和 ETL 都是用于集成数据的重要技术。用户可以通过复制功能实时地汇集分散在各个部门的多个数据库的数据，企业能够获取最新的数据以进行分析。用户可以使用基于 EII 的

数据联邦来编写应用程序，无需拷贝数据即可访问来自异构数据源的最新数据。最后，ETL 还提供抽取、转换和装载数据的技术，从而满足用户提取和转换大量数据的需要。Sybase 数据集成套件通过提供各种数据集成技术来满足典型数据集成项目的需要。另外，Sybase DI 给用户提供了处理非结构化内容所需要的搜索功能，以及无需轮询即可对数据库事件做出实时响应的能力。

4.4 本章小结

在本章内，我们着重为大家介绍了数据集成技术的概念、发展历程以及 Oracle、Microsoft、IBM、Sybase 等几家大公司在数据集成领域的解决方案与工具，希望通过本章的内容介绍，读者能够对数据集成技术有一个较为清楚明晰的认识，为学习后面章节的内容打下基础。

第 5 章

数据查询、分析与建模技术

随着当今各类数据的数据量的爆炸式增长，海量数据的相关研究也开始备受关注。海量数据的特点是数据规模海量、数据价值密度低、数据类型多样。但在数据的分析过程中，要求能够高速有效地对数据进行处理，因此对数据的查询和分析提出了很高的要求。本章主要介绍海量数据的查询、分析与建模技术。首先通过对数据查询、分析和建模的各项技术以及发展进行相应的介绍，建立数据查询、分析和建模的概念。随后对世界上可以有效地进行数据查询、分析和建模的工具进行介绍，使得读者对于海量数据查询、分析与建模有更进一步的了解。

5.1 数据查询、分析与建模技术介绍

本节主要对数据查询、分析与建模技术进行介绍。数据查询即为数据检索，针对海量数据，对用户或企业所要求的数据进行查询，并返回结果。数据分析即对海量数据进行分析处理，总结数据中所蕴含的各种规律，使得海量数据的价值得以体现。数据建模是根据数据的特点对数据的存储形式及结构进行建模，使得数据的存储能够更好地适应需求，加快数据的处理速度。

5.1.1 数据查询

海量数据查询技术即为海量数据检索技术，对数据进行检索，方便用户快速有效地找到用户需要的数据。

1. 海量数据查询的发展

海量数据是指巨大的、浩瀚的数据。随着信息化程度的提高，每一个现代人无不在充分感受数据应用带来的巨大变化。数据已由原始的形式逐渐丰富为图像、声音、视频等。现在，在许多行业中都需要操作海量数据，如电商、物联网以及其他部门。这些部门的数据至少达到了 TB 级。

在数字时代，海量数据的发展需要解决三个问题：海量数据的存储、海量数据的搜索以及商业智能。

（1）海量数据的存储

数据中蕴藏着企业的财富，但由于数据的增长速度太快，因此首先要解决的是海量数据的存储问题。这种存储是企业信息化的基础架构，更多地定位在硬件方面。随着数据量的激增，客观上逼迫企业必须实施海量存储的解决方案，海量存储设备的不断更新为解决此问题提供了可能。海量数据的存储相关技术内容已经在前几章介绍过，这里不再赘述。

（2）海量数据的搜索

如今，海量数据的存储已经提出了多种方案，包括众多云计算的开源项目。海量数据的搜索已经成为制约信息化进一步深化的瓶颈。目前具有一定信息化程度的企业都有自己的数据库，而利用数据库都可以实现查询。这就引出了一个“时间成本”的问题。当数据量达到一定级别，查询条件达到一定数量，同时有多人查询时，要从一个数据库中找到自己需要的数据通常就会花费较长的时间，如果每天有大量时间花在数据库的搜索上，那就将造成高额的时间成本。而如果要提高数据库的查询速度，就必须对数据库进行大量的索引配置并对硬件进行大幅度升级，这样又会造成设备成本的提高。因此，从应用的角度看，迫切需要一些新技术来解决海量数据的快速搜索问题。而海量数据存储的复杂性为海量数据的搜索提高了难度。实际搜索过程中要进行海量数据搜索不得不面对复杂的存储环境：庞大的网络环境，多样的存储介质，不同类型、不同格式的存储平台等。

（3）商业智能

商业智能是指一种能力，通过智能地使用企业的数据财产来制定更好的商务决策。各种企业的决策人员以企业中的数据仓库（Data Warehouse）为本，经由各式各样的查询分析工具（Query/Report Tools）、联机分析处理（OLAP）工具，或是数据挖掘（Data Mining）工具加上决策规划人员的行业知识（Industry Knowledge），从数据仓库中获得有利的信息，进而帮助企业获利，提高生产力与竞争力。

2. 海量数据查询的难点

- 能干扰和破坏企业现有的数据结构和常规业务流程。
- 必须实现多约束条件、多数据源、多数据格式、多人同时的高效搜索。
- 必须实现对硬件成本的良好控制。
- 简化相关开发和性能优化过程。

从国内企业目前的信息化现状，特别是数据应用的状况来看，海量数据查询已经是信息化进一步深入的瓶颈，也是未来商业智能发展的入门关。推动海量数据查询技术已经不是靠少数厂商销售他们的软件、硬件能解决的问题了，用户的参与和认可已经成为必需。要发展海量查询技术，可以借鉴其他重要技术的推广和普及方式，比如开放源代码、建立开源社区。开放源代码之所以能迅速普及，要归功于开放源代码协会。这些开源组织和社区是理论与技术发展的重要途径，也是目前技术人员最为青睐的发展方式。

5.1.2 数据分析

目前，社交网站、电子商务等网络服务的迅速发展，使得网络服务及网络信息规模裂变式增长，这样就会对大规模数据的处理带来了很大的挑战。金融业、零售业、医疗、电信、航空等领域也会产生大量的数据，在数据挖掘中如何处理海量数据，提高挖掘质量和效率，是迫切需要解决的问题。数据固有的记录历史信息的能力，使得企业认识到，大量数据中，尤其是历史数据中，是隐藏着许多有价值的东西的。通过对历史数据的分析，能够对现在和未来的业务发展有很大的帮助。这种分析需要两点的支持，一是对海量数据的规整和处理，数据的量越多，数据的种类越丰富，其提供的结果越准确、越详细。二是有数据统计分析的方法，根据分析业务内容的不同，使用的分析方法也会有所不同，常用的几种分析方法包括：分类、聚合、关联等。

数据分析最重要的领域为数据挖掘。针对海量数据的增长速度，许多国内外从事海量数据挖掘、知识发现领域的相关人士进行了深入的研究。海量数据的存储和处理能力本身就对数据挖掘或机器学习提出了很高的要求，Google 在这方面做的工作很有意义。Google 公司提出的 MapReduce 是可以在大型计算机集群上对海量数据进行并发处理的一种框架模型。它首先通过设定一个 Map 函数把输入数据变换成相应的键-值对，然后通过自定义的 Reduce 函数聚集起来具有同样键的值，并输出结果。现实世界中大都可以用此模型来表示对海量数据的处理。另外，并行数据库是数据库技术与并行技术结合的产物，并被视为一种高性能的数据库系统，它能大大提高关系型数据库中处理海量数据的效率。

5.1.3 数据建模

数据模型是对信息系统中客观事物及其联系的数据描述，它是复杂的数据关系之间的一个整体逻辑结构图。数据模型不但提供了整个组织藉以收集数据的基础，它还与组织中其他模型一起，精确恰当地记录业务需求，并支持信息系统不断地发展和完善，以满足不断变化的业务需求。对于任何一个信息系统来说，数据模型都是它的核心和灵魂。

数据建模是一种用于定义和分析数据的要求和其需要的相应支持的信息系统的过程。因此，数据建模的过程中，涉及的专业数据建模工作，与企业的利益和用户的信息系统密切相关。

从需求到实际的数据库，有三种不同的类型。用于信息系统的数据模型作为一个概念数据模型，本质上是一组记录数据要求的最初的规范技术。数据首先适合企业的最初要求，然后被转变为一个逻辑数据模型，该模型可以在数据库中的数据结构概念模型中实现。一个概念数据模型的实现可能需要多个逻辑数据模型。数据建模中的最后一步是确定逻辑数据模型到物理数据模型中对数据、访问、性能和存储的具体要求。数据建模定义的不只是数据元素，也包括它们的结构和它们之间的关系。

5.2 数据查询、分析与建模技术研究现状

本节主要对数据查询、分析与建模技术的研究现状进行探讨。首先介绍的是并行处理技术，海量数据的处理仅仅依靠单一的处理器进行处理是远远不够的，而海量数据处理中目前最为突出的技术即为并行处理技术，众多的海量数据处理工具都用到了并行处理的理念。并行处理技术使得大规模的海量数据处理变得快速高效。之后针对数据查询、分析与建模给出了多种技术的介绍。着重对 OLAP 以及数据挖掘技术进行了介绍。

5.2.1 并行处理

并行处理是计算机系统中能同时执行两个或两个以上任务的一种计算方法。处理机可同时工作于同一程序的不同方面。并行处理的主要目的是节省大型和复杂问题的解决时间。为使用并行处理，首先需要对程序进行并行化处理，也就是说将工作各部分分配到不同处理机中。正是这种使用多处理机来实现的方式，促成了所谓的“并行计算机”的出现。

当前流行的高性能并行机体系结构分为 4 类：对称多处理共享存储并行机（Symmetric Multi-Processing，SMP）、分布共享存储并行机（Distributed Shared Memory，DSM）、大规模并行计算机（Massively Parallel Processing，MPP）和工作站（微机）机群（Cluster Of Workstation，COW）。随着科技的进步和发展，结构分析问题正日益向非线性、大规模方向发展，MPP 已是当今超级计算机的主要结构之一。

MPP 提供了一种进行系统扩展的方式，它由多个 SMP（Symmetric Multi-Processor）服务器通过一定的节点互联网络进行连接，协同工作，完成相同的任务，从用户的角度来看是一个服务器系统。其基本特征是由多个 SMP 服务器（每个 SMP 服务器称节点）通过节点互联网络连接而成，每个节点只访问自己的本地资源（内存、存储等），是一种完全无共享（Share Nothing）结构，因而扩展能力最好，理论上其扩展无限制。目前业界对节点互联网络暂无标准，如 NCR 的 Bynet，IBM 的 SPSwitch，它们都采用了不同的内部实现机制。但节点互联网仅供 MPP 服务器内部使用，对用户而言是透明的。

在 MPP 系统中，每个 SMP 节点也可以运行自己的操作系统、数据库等。每个节点内的 CPU 不能访问另一个节点的内存。节点之间的信息交互是通过节点互联网络实现的，这个过程一般称为数据重分配（Data Redistribution）。

MPP 服务器需要一种复杂的机制来调度和平衡各个节点的负载和并行处理过程。目前一些基于 MPP 技术的服务器往往通过系统级软件（如数据库）来屏蔽这种复杂性。举例来说，NCR 的 Teradata 就是基于 MPP 技术的一个关系数据库软件，基于此数据库来开发应用时，不管后台服务器由多少个节点组成，开发人员所面对的都是同一个数据库系统，而不需要考虑如何调度其中某几个节点的负载。

使用 MPP 使多台计算机同时处理同一问题的方法主要有两种：

（1）无共享并行体系结构

无共享（shared nothing）体系结构意味着每台计算机都有它自己的 CPU、内存和磁盘。计算机通过高速互联网络被连接在一起，当处理查询时，每个节点处理其本地表中的行，然后将节点的部分结果回传给协调程序节点。协调程序将来自所有节点的所有结果合并成最终结果集。节点不一定是独立的计算机，在单个计算机上可以存在多个分区。如图 5.1 所示为无共享并行体系结构。

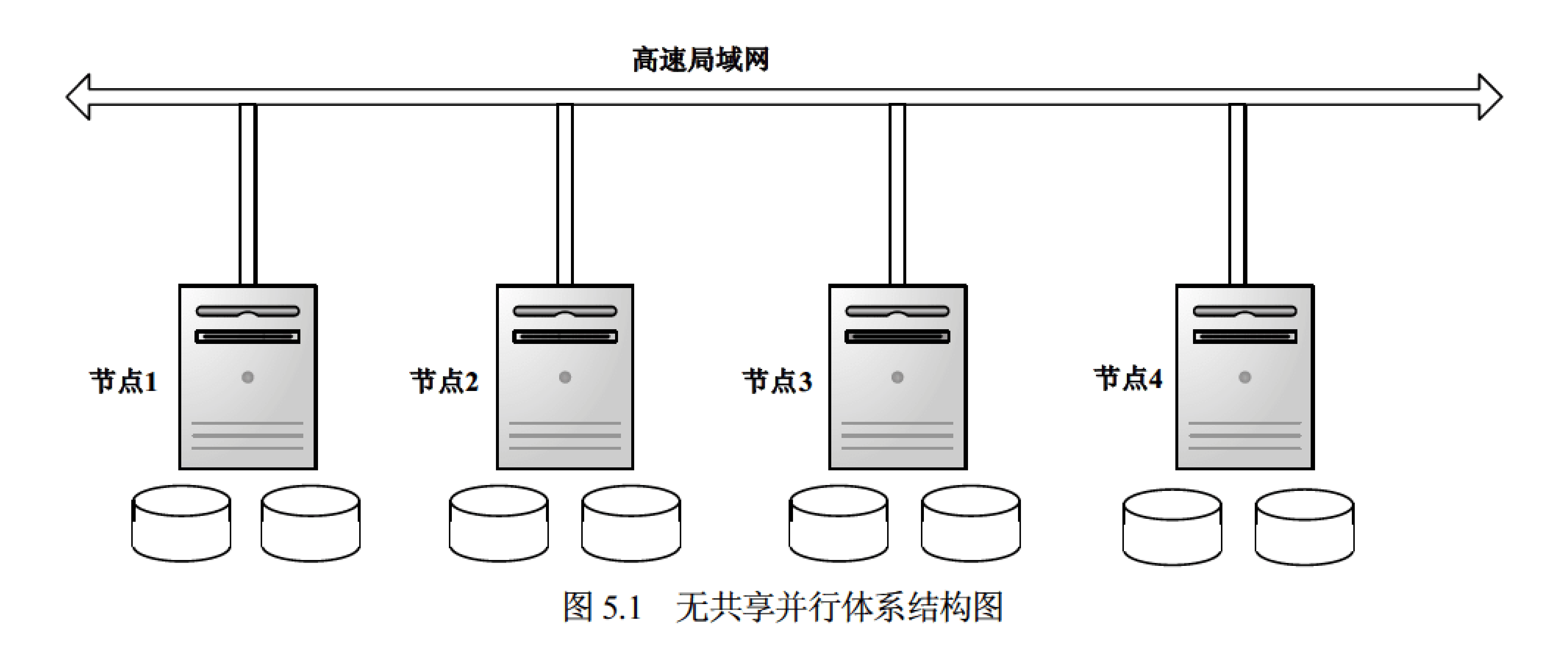


图 5.1　无共享并行体系结构图

（2）分布式锁（共享磁盘）并行体系结构

共享磁盘体系结构使用锁管理器在计算机之间做出仲裁，这些计算机都访问一个公共磁盘池。这种体系结构在原理上是有缺陷的，因为当节点数增加时，群集会过多集中在锁请求中。如图 5.2 所示为共享磁盘并行体系结构。

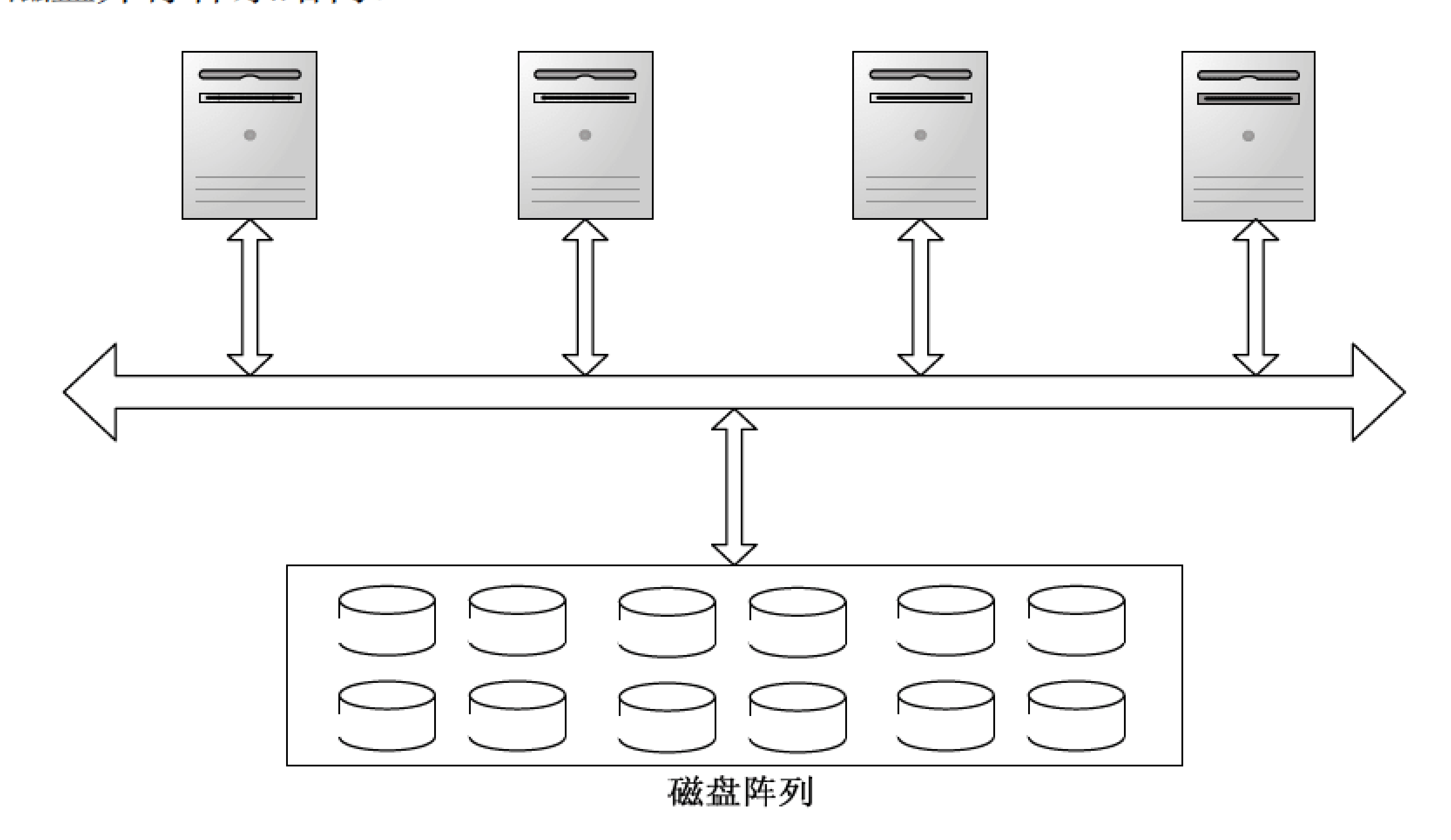


图 5.2　共享磁盘并行体系结构图

该环境中的数据库访问要求每个节点都在共享磁盘上请求一块数据。如果另一个节点已经锁定该数据准备更新，那么正在请求的节点必须等待正在更新的节点完成。这种环境容易发生“死锁”的情况。节点 1 锁定资源 A 并请求资源 B。节点 2 锁定资源 B 并请求对资源 A 的锁定。

这两个节点都不释放它们的锁，所以它们一直等待下去。

5.2.2 海量数据查询与搜索

在面对海量数据时，用户关心的是如何从其中查询出对自己有价值的信息而非海量数据本身，如何使查询海量数据的性能更为高效是目前国内外数据库系统研究的热点问题。在海量数据查询优化策略有：

- 查询语句优化。对查询语句进行变换以减少语句执行开销。
- 规则优化。根据启发式规则选择执行策略。但两种方式存在不足：当数据量超出系统软硬件处理能力时，通过优化语句很难提升查询性能。
- 物理优化。选择合适的存储策略进行的优化，但是对语句的执行效率考虑不足。
- 代价估算优化。对已经存在的优化策略进行代价估算，选择最小的执行代价策略。这种方式的问题在于计算最小执行代价耗费时间过多且实用性不高。

目前的海量数据查询性能低下大都是由于数据的规模超出系统的软硬件处理能力。

这里讨论利用多数据库中间件插件技术在存储海量数据时将数据划分存储到多个自治的数据库中，在一维上降低数据规模并优化查询语句；在本地数据库中运用表分区技术，将海量数据划分存储到多个表分区中，通过增加维度降低海量数据的规模；利用分表措施将分区表分成多个子表，再次降低海量数据的规模。通过对海量数据的三维划分、优化查询语句以及降低数据的扫描规模提高了海量数据的查询性能。

1. 基于数据划分的海量数据查询

数据划分是指按照某种规则将数据分布到特定范围内，使得在对数据进行查询时系统并行处理能力提高，以此降低查询的响应时间，提高数据库的查询性能。数据划分对于能否充分利用系统的 CPU 和带宽资源，减少通信开销，平衡系统负载和减少计算量，最佳的发挥并行性和系统性能至关重要。

（1）多数据库并行处理技术

多数据库并行处理技术是以中间件为技术支撑，对海量数据进行合理存储，高效查询的一种技术。多数据库并行处理结构如图 5.3 所示。

当用户提出加载请求时，通过负载均衡系统将请求均衡的分发给并行加载服务，并行加载服务首先读取全局数据字典中的元数据，通过数据划分器和表加载器将数据加载到底层数据库中，当用户提出查询服务时，系统将请求发给并行查询服务，并行查询服务首先读取全局数据字典用于获得多数据库的配置信息，然后通过查询语句改写服务优化查询语句并发送给查询服务器，用以完成数据的查询功能。

多数据库并行查询技术通过中间层组件对查询语句进行分析、优化，根据分析的结果将查询分解或者复制为多个等价的子查询，将多个子查询语句在相应的数据库节点上执行，它降低了每个本地数据库的查询规模，并实现查询的本地化并行查询，提高了数据的查询效率。

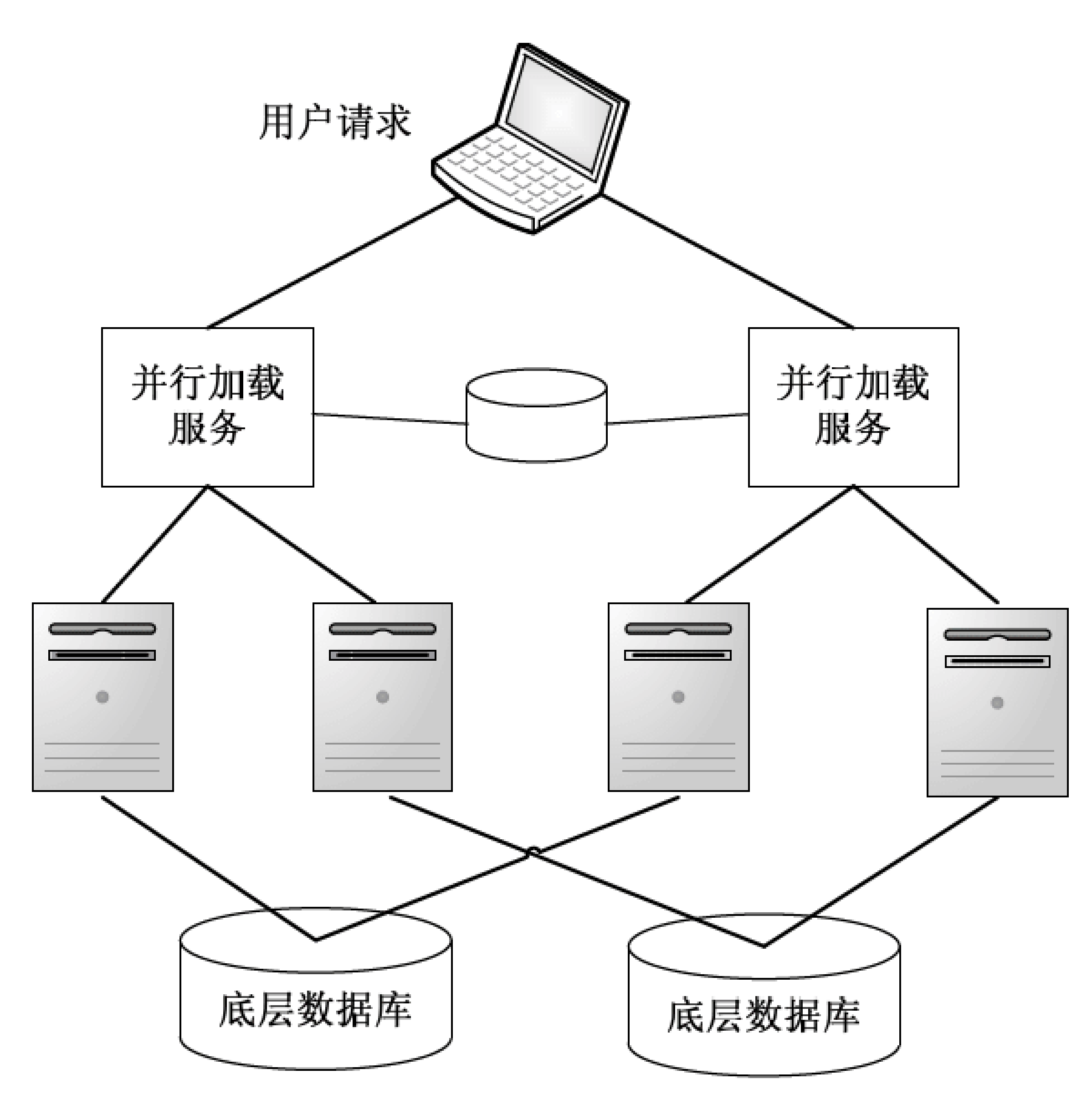


图 5.3　多数据库并行处理结构图

（2）表分区

针对底层数据库数据查询规模大的问题，对其特定业务的查询条件属性做表分区处理，以提高查询的性能。经表分区后，数据在数据库中按一定的规律存放。

当进行数据查询时，系统只需扫描存放数据的响应表分区，因而查询扫描规模可以大大降低，数据查询性能也会得到相应的提高。 当分区的规模较大时，在海量数据库中无法有效降低数据规模，这样的数据划分无意义；当用户分区规模较小时，在进行多数据查询时，会涉及多个表分区，查询性能也难以提高。因此表分区的粒度需要优先考虑两方面：一是数据的存储规模；二是用户的查询粒度。

（3）分表技术

● *原理*

在分区表的基础上，将表根据所查找的属性字段划分成为多个子表，利用数据库的 Union ALL 视图将需要查询的子表动态地合并起来，使对原表的查询转化为对视图或者子表的查询，以此降低数据查询的扫描规模。

● *算法*

输入：业务层提交给中间层的查询语句。

输出：语句改写后的查询语句。

流程：

a.将业务层提交的查询语句提交给语法分析器；通过语法分析器截取 where 子句后的条件表达式。

b.分析条件表达式中是否含有与分表字段有关的条件。当不包含分表字段时返回原语句，算法结束；否则提取该字段。

c.根据提取的字段分析其涉及的数据范围，找到相对应的视图或者子表。

d.将原语句中 from 子句后的原表改为 c 得到的视图或者子表，此时得到查询语句 a。

e.去除 a 中与分表有关字段的条件，返回改写后的查询语句，算法结束。

2. 基于搜索引擎的海量数据查询

众所周知，搜索引擎是从互联网上搜寻信息的重要工具。随着互联网规模的不断增大，网上信息量的不断增长，搜索引擎的作用越来越明显。搜索引擎的技术基础是全文检索技术。搜索引擎从实现技术上一般可以分为三类：目录型、通用型和元搜索引擎。目录型搜索引擎通过人工编辑的网站分类目录提供导航和网站搜索；元搜索引擎自身不维护数据，而对多个通用型搜索引擎的检索结果进行融合与重新排序，期望为用户提供更全和更好的检索服务；通用型搜索引擎通过抓取 Web 网页，对抓取的网页数据处理、索引，为用户提供检索服务。

搜索引擎由搜索器、索引器、检索器和用户接口 4 个部分组成。

（1）搜索器（crawl）

搜索引擎使用一种名为“网络机器人”或“网络蜘蛛”的软件，遍历 Web 空间，扫描一定 IP 地址范围内的网站，并沿着网络上的链接从一个网页到另一个网页收集网页数据。它为保证收集的数据最新，还会回访已收集过的网页。

（2）索引器（index）

索引器的功能是理解搜索器所搜索的信息，从中抽取出索引项，用于表示文档以及生成文档库的索引表。索引器可以使用集中式索引算法或分布式索引算法。当数据量很大时，必须实现即时索引，否则跟不上信息量急剧增加的速度。索引算法对索引器的性能（如大规模峰值查询时的响应速度）有很大的影响。一个搜索引擎的有效性在很大程度上取决于索引器的质量。

（3）检索器（search）

检索器的功能是根据用户的查询在索引库中快速检索出所需内容。

随着互联网的迅猛发展，网络信息的增加，用户要在海量的数据信息里查找所需信息，如果没有强有力的信息检索和分析工具几乎是不可能的。当前使用比较普遍的检索系统能部分解决资源发现的问题，但随着网络数据量急剧膨胀，传统的集中式信息检索方法在搜索性能上已经无法满足用户的要求。当信息的检索规模达到一定的程度时，必然要采用分布式的方法，以提高系统性能。

分布式数据检索技术的提出对信息检索领域具有极其重大的意义。与传统的信息检索技术相比，使用分布式数据检索技术的检索系统在检索效率上有了大幅的提高。而如何进一步在分布式数据检索技术的基础上提高其检索效率，是一个值得研究的课题。最优搜索理论是二战时期发展起来的学科，是计算统筹学的分支，它研究在有限的资源约束条件下，如何分配资源使得成功搜索到目标的可能性最大或者资源的消耗最小。利用最优搜索理论对分布式数据检索进行优化，将

提高系统的检索质量。

5.2.3 数据分析中的 OLAP 与数据挖掘技术

随着数据库技术的广泛应用，企业信息系统产生了大量的数据，如何从这些海量数据中提取对企业决策分析有用的信息成为企业决策管理人员所面临的重要难题。传统的企业数据库系统（面向企业事务处理的信息系统）即联机事务处理系统（On-line Transaction Processing，OLTP）作为事务管理的手段，主要用于企业事务处理，但它对分析处理的支持一直不能令人满意。因此，人们逐渐尝试对 OLTP 数据库中的数据进行再加工，形成一个综合的、面向分析的、更好的支持决策制定的决策支持系统（Decision Support System，DSS）。企业目前的信息系统的数据一般由 RDBMS 管理，但决策分析数据库和联机事务处理数据库在数据来源、数据内容、数据模式、服务对象、访问方式、事务管理乃至物理存储等当面都有不同的特点和要求，因此直接在联机事务处理数据库上建立 DSS 是不合适的。数据仓库（Data Warehouse）技术就是在这样的背景下发展起来的。数据仓库的概念提出于 20 世纪 80 年代中期，20 世纪 90 年代，数据仓库已从早期的探索阶段走向试用阶段。业界公认的数据仓库概念的创始人 W.H.Inmon 在《Building the Data Warehouse》一书中对数据仓库的定义是："数据仓库是支持管理决策过程的、面向主题的、集成的、随时间变化的持久的数据集合。"构建数据仓库的过程就是根据预先设计好的逻辑模式从分布在企业内部各处的联机事务处理数据库中提取数据，并经过必要的转换最终形成全企业统一模式的数据的过程。当前数据仓库的核心（即数据仓库数据库）是 RDBMS 管理下的一个数据库系统。数据仓库中数据量巨大，为了提高性能，RDBMS 一般也采取一些提高效率的措施，比如采用并行处理结构、新的数据组织、查询策略、索引技术等。

联机分析处理（On-line Analytical Processing，OLAP）的概念最早由关系数据库之父 E. F. Codd 于 1993 年提出。Codd 认为联机事务处理（OLTP）已经不能满足终端用户对数据库查询分析的要求，SQL 对大数据库的简单查询也不能满足用户分析的需求。用户的决策分析需要对关系数据库进行大量计算才能得到结果，而查询的结果并不能满足决策者提出的需求。因此，Codd 提出了多维数据库和多维分析的概念，即 OLAP。

OLAP 技术与关系数据库结合已经成为一种趋势，像 Oracle 公司和 Microsoft 公司都在这方面做了相应的工作。Oracle 公司从 Oracle 9i 开始，在其数据库管理系统中加入了 OLAP 功能，实现了把多维数据库保存在关系型数据库当中的承诺。成为业界第一个实现这种功能的产品。OLAP 技术终于可以和关系型数据库完全融合在一起了，在其以后的数据库版本中，这个特性更是发挥得淋漓尽致。Microsoft 公司的 Analysis Services 也是对数据仓库和 OLAP 的支持。

1. OLAP 的定义和特征

联机分析处理是一类软件技术，它使分析人员、管理人员通过对信息的多种可能的观察角度进行快速、一致和交互性的存取以获得对信息的深入理解，这些信息从原始数据转换而来，反映了用户所能理解的企业的真实的"维"。目前所指的联机分析处理，是对数据的一系列交互的查询过程，这些查询过程要求对数据进行多层次、多阶段的分析处理，获得高度归纳的信息。归纳信

息要从最底层的明细数据开始，经过多个层次、多个阶段的数据处理，包括数据汇总、整理、归纳、排除奇异数据样本等环节，最终得到用户所需要的经过归纳抽象的信息。OLAP 力图处理数据仓库中浩瀚如烟的数据，并将之转化为有用的信息，从而实现对数据的归纳、分析和处理，帮助企业完成决策。OLAP 支持最终用户进行动态多维分析，其中包括跨维、在不同的层次之间跨成员的计算和建模。

20 世纪 80 年代，E．F．Codd 提出了 OLAP 数据库的十二条准则，被广泛作为管理公司日常事务的数据库标准。1993 年，E. F. Codd 规定了 OLAP 的如下十二条准则：

- OLAP 模型必须提供多维概念视图
- 透明性准则
- 存取能力准则
- 稳定的报表性能
- 智能化的客户／服务器体系结构
- 维的等价性和通用性
- 动态稀疏数据矩阵处理
- 支持多用户
- 支持非限定的交叉维操作
- 能直接访问数据
- 具有随机灵活的报表机制
- 提供不受限制的维和聚集级别

然而 E. F. Codd 提出的 OLAP 的十二条准则并没有得到广泛的承认，大多数专家认为 OLAP 并不需要遵守这些准则，准则只是提供了一种数据技术的观点，而不是基准。但是术语 OLAP 被用来很好地描述为推动公司决策制定、分析而设计的数据库。OLAP 通常是指使得数据仓库的数据能被很容易被访问的工具。

OLAP 技术主要有两个特点：一是在线性（On-line），表现为对用户请求的快速响应和交互操作；二是多维分析（Multi-Analysis），这也是 OLAP 技术的核心所在。

OLAP 的最显著特征是它能提供数据的多维概念视图（Multi-Dimensional）。在 OLAP 数据模型中，多维信息被抽象为一个立方体（Cube），它包括维（Dimension）和度量（Measure）。维就是我们所说的观察角度，而度量则是上面说的指标值。多维结构是 OLAP 的核心，OLAP 展现在用户面前的就是一幅幅多维视图。这些多维视图能使最终用户从多角度、多侧面、多层次直观地考察数据仓库中的数据，从而深入地理解包含在数据中的信息及其内涵。以多维视图的形式把数据提供给用户，既迎合了人的思维模式又减少了概念上的混淆，同时降低了出现错误解释的可能性。

OLAP 的第二个特性是它能快速响应用户的分析需求．一般认为 OLAP 系统应在几秒内对用户的分析请求做出响应。如果终端用户在 30 秒内没有得到系统响应就会变得不耐烦，因而可能失去分析主线索，影响分析质量。对于大量的数据分析要达到这个速度并不容易，因此就更需要一些技术上的支持，如专门的数据存储格式、大量的事先运算、特别的硬件设计等。

OLAP 的第三个特征是它的分析功能（Analysis）。OLAP 系统应该能处理与应用有关的任何逻辑分析和统计分析。尽管系统可以事先编程，但并不意味着系统定义了所有的应用。在应用 OLAP 的过程中，用户无需编程就可以定义新的专门计算，将其作为分析的一部分，且以用户所希望的方式给出报告。用户可在 OLAP 平台上进行数据分析，也可连接到其他外部分析工具上，如时间序列分析工具、成本分析工具、意外报警、数据挖掘等。OLAP 的基本分析操作有切片（Slice）、切块（Dice）、钻取（Drill-down）、卷取（Drill-up）及旋转（Rotate）。

OLAP 的第四个特征是它的信息性（Information）。无论数据量有多大，也不管数据存储在何处，OLAP 系统应能及时获得信息，并且管理大容量信息。这里有许多因素需要考虑，如数据的可复制性、可利用的磁盘空间、OLAP 产品的性能以及与数据仓库的结合度等。

OLAP 是针对特定问题的联机数据访问和分析。通过对信息的很多种可能的观察形式进行快速、稳定一致和交互性的存取，允许管理决策人员对数据进行深入观察。为了对 OLAP 技术有更深入的了解，以下主要介绍在 OLAP 中常用的一些基本概念。

（1）变量

变量是数据的实际意义，即描述数据“是什么”。一般情况下，变量总是一个数值度量指标，例如人数、单价、销售量等都是变量。

（2）维

维是人们观察数据的特定角度。例如，企业常常关心产品销售数据随着时间推移而产生的变化情况，这时是从时间的角度来观察产品的销售，所以时间是一个维（时间维）。企业也时常关心自己的产品在不同地区的销售分布情况，这时是从地理分布的角度来观察产品的销售，所以地理分布也是一个维（地理维）。其他还有如产品维、顾客维等。

（3）维的层次

人们观察数据的某个特定角度（即某个维）还可以存在细节程度不同的多个描述方面，我们称这多个描述方面为维的层次。一个维往往具有多个层次，例如描述时间维时，可以从日期、月份、季度、年等不同层次来描述，那么日期、月份、季度、年等就是时间维的层次；同样，城市、地区、国家等构成了地理维的层次。

（4）维成员

维的一个取值称为该维的一个维成员。如果一个维是多层次的，那么该维的维成员是由各个不同维层次的取值组合而成的。

（5）多维数组

一个多维数组可以表示为：（维 1，维 2，…维 n，变量）。

（6）数据单元

多维数组的取值称为数据单元。当多维数组的每个维都选中一个维成员，这些维成员的组合就唯一确定了一个变量的值。那么数据单元就可以表示为（维 1 维成员，维 2 维成员，…，维 n 维成员，变量的值）。

2. 数据挖掘

（1）数据挖掘的定义

数据挖掘（Data Mining）是指基于一定业务目标从海量数据中挖取潜在的、合理的并能被人理解的模式的高级处理过程。与传统的数据分析最大本质区别是数据分析所得到的信息具有先前未知、有效和实用三个特征，即数据挖掘是发现那些不能靠直觉发现的信息或知识，甚至违背直觉的信息或知识，挖掘出来的信息越出乎意料越有价值。

（2）数据挖掘的特点

- 数据：挖掘行为的数据来源，是关于主题的集合，是进行挖掘和知识发现的原始材料。
- 新颖：数据挖掘和知识发现的模式应该是新颖的，它可以通过当前得到的数据和同期相比得到的数据的新颖程度，或者通过知识发现的内容和原先发现的内容相比的新颖程度来判定模式的新颖。
- 隐含应用性：提取的数据应该是对人们有价值的信息，即按照商业主题为对象的数据挖掘具有经济价值或实用价值。

（3）数据挖掘的分类

数据挖掘可按数据库类型、挖掘对象、挖掘任务、挖掘方法与技术以及应用等几方面进行分类。

数据挖掘最开始是从在关系数据库中挖掘知识发展起来的，随着数据库类型的不断增加，现有：关系数据挖掘、模糊数据挖掘、历史数据挖掘、空间数据挖掘等多种不同数据库的数据挖掘类型。

按挖掘的对象分，除了数据库数据挖掘外，还有文本数据挖据、多媒体数据挖掘、Web 数据挖掘。

按挖掘任务分类有关联规则挖掘、序列模式挖掘、聚类数据挖掘、分类数据挖掘、偏差数据挖掘和预测数据挖掘等类型。各类数据挖掘任务不同，采用的方法和技术也将会不同。

按挖掘的方法和技术分类有归纳学习类、仿生物技术类、公式发现类、统计分析类、模糊数学类、可视化技术类等。

（4）数据挖掘的基本任务

数据挖掘的任务主要是关联分析、聚类分析、分类、预测、时序模式和偏差分析等。

- 关联分析（Association Analysis）

关联规则挖掘由 Rakesh Apwal 等人首先提出。两个或两个以上变量的取值之间存在的规律性称为关联。数据关联是数据库中存在的一类重要的、可被发现的知识。关联分为简单关联、时序关联和因果关联。关联分析的目的是找出数据库中隐藏的关联网。一般用支持度和可信度两个阀值来度量关联规则的相关性，还不断引入兴趣度、相关性等参数，使得所挖掘的规则更符合需求。

- 聚类分析（Clustering）

聚类是把数据按照相似性归纳成若干类别，同一类中的数据彼此相似，不同类中的数据相

异。聚类分析可以建立宏观的概念，发现数据的分布模式，以及可能的数据属性之间的相互关系。

- 分类（Classification）

分类就是找出一个类别的概念描述，它代表了这类数据的整体信息，即该类的内涵描述，并用这种描述来构造模型，一般用规则或决策树模式表示。分类是利用训练数据集通过一定的算法而求得分类规则。分类可被用于规则描述和预测。

- 预测（Predication）

预测是利用历史数据找出变化规律，建立模型，并由此模型对未来数据的种类及特征进行预测。预测关心的是精度和不确定性，通常用预测方差来度量。

- 时序模式（Time-series Pattern）

时序模式是指通过时间序列搜索出的重复发生概率较高的模式。与回归一样，它也是用已知的数据预测未来的值，但这些数据的区别是变量所处时间的不同。

- 偏差分析（deviation）

在偏差中包括很多有用的知识，数据库中的数据存在很多异常情况，发现数据库中数据存在的异常情况是非常重要的。偏差检验的基本方法就是寻找观察结果与参照之间的差别。

（5）数据挖掘的基本技术

- 统计学

统计学虽然是一门“古老的”学科，但它依然是最基本的数据挖掘技术，特别是多元统计分析，如判别分析、主成分分析、因子分析、相关分析、多元回归分析等。

- 聚类分析和模式识别

聚类分析主要是根据事物的特征对其进行聚类或分类，即所谓物以类聚，以期从中发现规律和典型模式。这类技术是数据挖掘的最重要的技术之一。除传统的基于多元统计分析的聚类方法外，近些年来模糊聚类和神经网络聚类方法也有了长足的发展。

- 决策树分类技术

决策树分类是根据不同的重要特征，以树型结构表示分类或决策集合，从而产生规则和发现规律。

- 人工神经网络和遗传基因算法

人工神经网络是一个迅速发展的前沿研究领域，对计算机科学、人工智能、认知科学以及信息技术等产生了重要而深远的影响，而它在数据挖掘中也扮演着非常重要的角色。人工神经网络可通过示例学习，形成描述复杂非线性系统的非线性函数，这实际上是得到了客观规律的定量描述，有

了这个基础，预测的难题就会迎刃而解。目前在数据挖掘中，最常使用的两种神经网络是 BP 网络和 RBF 网络。不过，由于人工神经网络还是一个新兴学科，一些重要的理论问题尚未解决。

- 规则归纳

规则归纳相对来讲是数据挖掘特有的技术。它指的是在大型数据库或数据仓库中搜索和挖掘以往不知道的规则和规律，大致包括以下形式：IF … THEN …

- 可视化技术

可视化技术是数据挖掘不可忽视的辅助技术。数据挖掘通常会涉及较复杂的数学方法和信息技术，为了方便用户理解和使用这类技术，必须借助图形、图像、动画等手段形象地指导操作、引导挖掘和表达结果等，否则很难推广普及数据挖掘技术。

（6）数据挖掘技术实施的步骤

数据挖掘的过程可以分为 6 个步骤。

- 理解业务：从商业的角度理解项目目标和需求，将其转换成一种数据挖掘的问题定义，设计出达到目标的一个初步计划。
- 理解数据：收集初步的数据，进行各种熟悉数据的活动。包括数据描述、数据探索和数据质量验证等。
- 准备数据：将最初的原始数据构造成最终适合建模工具处理的数据集。包括表、记录和属性的选择，数据转换和数据清理等。
- 建模：选择和应用各种建模技术，并对其参数进行优化。
- 模型评估：对模型进行较为彻底的评价，并检查构建模型的每个步骤，确认其是否真正实现了预定的商业目的。
- 模型部署：创建完模型并不意味着项目的结束，即使模型的目的是为了增进对数据的了解，所获得的知识也要用一种用户可以使用的方式来组织和表示。通常要将活动模型应用到决策制订的过程中去。该阶段可以简单到只生成一份报告，也可以复杂到在企业内实施一个可重复的数据挖掘过程，使其得到普遍承认。

（7）数据挖掘的应用

数据挖掘的应用领域非常广泛，未来世界的各个方面都需要数据挖掘，目前主要集中在如下领域。

- 商业领域

数据挖掘的商业领域主要是电子商务的出现和发展带动了数据挖掘的发展，客户关系 CRM 系统的繁荣和发展也带动了数据挖掘的发展，数据挖掘能发现商务客户的共性点和差异的/现实和未来预测的信息、必然的信息、独立或者关联的信息等，通过发掘这些信息知识能够归纳和总结用户的消费行为，如消费能力、消费需求、对产品的关注度、消费心理等，这些有价值的信息能够为管理者的决策提供依据和信息来源。在 CRM 即客户关系管理系统中可以根据需求对客户进

行分类，分析客户的消费能力、客户住址区域信息、客户购买产品的能力等。借助数据挖掘系统的相关工具如数据仓库、知识发现、数据决策分析等工具可以预测投资行情，如股票、期货，也可以用于分析电信、医疗行业，深入了解客户的喜好，调整营销策略和提高产品质量等。

- 科研领域

数据挖掘在科研领域的应用也非常广泛，主要是地理、医学、生物工程等方面。数据挖掘目前在远程教育的应用中比较成熟，老师可以根据学生的学习基础对学生的学习情况进行跟踪，根据学生的特点建立不同的教学方法库，动态调整教学方法和内容。数据挖掘在地理信息工程中的应用主要体现在空间数据的应用和研究上，空间数据挖掘可以理解为将数据挖掘和地理信息系统、遥感信息学、全球定位、模式识别等综合在一起的交叉学科的研究应用，空间应用也是现在数据挖掘中的一个重点和热门领域。数据挖掘在军事领域的应用对打赢信息化条件下的局部战争提供了很好的技术支持。

- Web 挖掘

随着 Internet 的迅猛发展，今天它已成为各行各业人们交流思想、获取信息的便利手段。但是这些信息缺乏结构化、组织的规整性。随意地散布在因特网的各个角落，这已成为这座世界性图书馆的一大遗憾。而今天因特网的规模在急剧地扩大。其上的信息量也在爆炸般地增长，这时人们若不去有意识地寻求弥补该缺憾的有效途径，在不久的将来人们将迷途于信息的汪洋中。数据挖掘在 Internet 上的应用包括：在搜索引擎上对文档进行自动分类、帮助寻找用户感兴趣的新闻以及利用数据挖掘设计一个电子新闻过滤系统。利用文本学习建立起该用户的趋向模型，当用户进入一份电子报纸的网页时，该系统就会根据学习所得的模型对其中的每一篇文章与用户的兴趣的接近程度进行打分排序，使用户最先看到他最感兴趣的新闻。

（8）数据挖掘的价值实现难点分析

数据挖掘是数据库中的知识发现，从知识发现到知识应用、再到价值评估是一条数据挖掘价值变现的过程，虽然数据挖掘重要性毋庸置疑，但事实上其在实现商业价值的道路上仍有较多困难。

- 知识发现

知识发现是这条路的始端，直接决定了最终价值的高度。挖掘的方法是通用的，但难度不在挖掘技术，而在于实施人员对数据业务的理解，在于数据的质量。实施人员必须清楚地知道数据回收的场景和原理，稍有沟通缺失，都会影响知识的质量度。

- 知识应用

发现了知识，只是迈出第一步，需要将相关的知识发现交给业务部门进行运营使用。不管是以甲方公司还是乙方公司的形式存在，难点在于语言的翻译转发。数据挖掘的语言形式是概率形式，类如“连续三天内在站内搜索超过 10 次，浏览搜索结果相关页面 20 次以上的用户最终购买概率为 42%”，因此需要实施人员深谙运营知识，将挖掘结果语言转化成运营结果语言，最终成

为友好的商业运营。应用的过程还需要及时跟踪、分析、调整，毕竟市场是多变的，分析与执行就像左脑和右脑，两者距离的远近，影响结果的优劣。

- 价值评估

数据挖掘的效果评估决定最终的话语和地位。从结果来看，如果结果有效，如何界定是知识有效还是执行有效；如果结果无效，如何界定是知识无效还是执行无效；如果知识有效，如何界定是通过挖掘发现还是已知发现。如果不能很清晰地界定，数据挖掘的存在价值都会大打折扣。曾经有个笑话，“通过海量数据发现，中国的 15~20 岁的男性网民最喜欢使用 QQ 即时通信工具”，这样的知识发现虽然是个笑话，但在现实行业里却是个不争的事实。数据挖掘的价值应当是显现的、直观的、令人信服的，不在于挖掘的技术多么高深，而在于整个体系的搭建和成果的展现，做得再好，看不到效果，也等于无效。

（9）国内外数据挖掘的研究现状

目前，数据挖掘的研究和应用已经引起人们的关注，学术界、实业界和政府部门越发重视数据挖掘的研究。以美国为核心的发达国家对数据挖掘的研究和应用取得了重大进展，在数据挖掘的研究领域，数据挖掘开展最早的也是美国，数据挖掘的核心研究还是在美国，作为具有全球影响力的 KDD（Knowledge Discovery in Database）学术会议从 1995 年到现在已经召开多次会议，其中大部分在美国召开，凸显了美国的重要作用和比重。 全球应用最为广泛的数据挖掘产品主要是美国研发生产出来 SPSS、SAS。

我国数据挖掘研究比美国晚，21 世纪才开始起步，2001~2003 年发表的这方面的论文比重很低，近年来该方面论文收录比重开始急剧上升，数据挖掘的研究越来越受到大家的重视，同时相关的 IT 公司也在研发这方面的产品，数据挖掘的人才培养也越来越受到高校、公司的重视。由此可见数据挖掘已成为一个热门的研究领域，将带动大量相关产业的发展。

（10）数据挖掘的发展趋势

经过多年的研究和发展，数据挖掘充分吸收了多门学科的最新研究成果，逐步形成了自己独特的风格特点，形成了独具特色的研究分支。但是，数据挖掘研究和应用仍具有巨大的挑战性，凡事都要有一个过程，数据挖掘的研究和发展也一样。

分析目前的研究和应用现状，数据挖掘在如下 6 个方面需要重点开展工作。

- 数据挖掘技术与特定商业逻辑的平滑集成问题

有效、显著的应用实例能够很好地证明数据挖掘和知识发现技术的广阔应用前景。数据挖掘过程很多关键课题已经嵌入了对行业或企业知识挖掘的约束与指导、商业逻辑等领域知识，这些领域知识将是数据挖掘与知识发现技术研究和应用的重点发展方向。

- 数据挖掘技术与特定数据存储类型的适应问题

数据挖掘的具体实现机制、目标定位、技术有效性会受到不同数据存储方式的影响。要想依靠单一通用的应用模式适合所有的数据存储方式去发现有效知识是不可能的。因此根据不同数据存储类型的特点进行针对性数据挖掘的研究是目前流行而且也是将来一段时间的主要问题之一。

● 大型数据的选择与规格化问题

在对大型数据集进行数据挖掘时，由于源数据库中的数据呈现动态变化，还有数据存在噪声、不确定性、信息丢失、信息冗余、数据分布稀疏等问题，必须进行挖掘前的预处理工作。对特定商业目标进行数据挖掘时，由于存在大量的数据，必须要选择性地利用，因此如何进行数据选择、规格化特定挖掘方法是无法回避的问题。

● 数据挖掘系统的构架与交互式挖掘技术

随着研究的发展，数据挖掘系统的基本构架已经逐步形成，过程已经趋于明朗，但是受其他因素的影响，在进行数据挖掘研究时，很多方面仍需要深入研究。由于数据挖掘是在大量的源数据集中发现潜在的、事先并不知道的知识，因此和用户交互式进行探索性挖掘是必然的。这种交互可能发生在数据挖掘的各个不同阶段，从不同角度或不同粒度进行交互。所以良好的交互式挖掘（Interaction Mining）也是数据挖掘系统成功的前提。

● 数据挖掘语言与系统的可视化问题

由于数据挖掘技术诞生较晚，加上其复杂的特点，在开发相应的数据挖掘操作语言时将会困难重重。可视化要求成为了信息处理系统的不可缺的技术。可视化挖掘除了要和良好的交互式技术结合外，还必须在挖掘结果或知识模式的可视化、挖掘过程的可视化以及可视化指导用户挖掘等方面进行探索和实践。数据的可视化从某种角度说起到了推动人们主动进行知识发现的作用，因为它可以是人们从对 KDD 的神秘感变成可以直观理解的知识和形象的过程。

● 数据挖掘理论与算法研究

经过研究，数据挖掘已经形成了独具特色的理论体系。但是，这绝不意味着挖掘理论的探索已经结束，而是给我们带来了更加丰富的理论研究课题，这些新理论面对实际应用目标进行数据挖掘时具有重要的指导作用，新理论的发展必然促进新的挖掘算法的产生，这些算法对扩展数据挖掘非常有效，因此，对数据挖掘理论和算法的探讨将是长期而艰巨的任务。

从上面的叙述可以看出，数据挖掘研究和探索的内容是极其丰富和具有挑战性的。

5.2.4　数据模型与数据建模方法

1. 数据库类型

随着关系数据库系统的使用越加广泛，也越加深入，有人指出关系数据库所具有的一些问题。为了解决关系数据库所固有的一些缺陷，很多研究人员投入到新的数据模型的研究上，出现了多种类型的数据系统，主要的数据库类型如下。

（1）对象数据库系统

它是数据库技术与面向对象程序设计方法相结合的产物，具有良好的扩展数据类型、支持任意复杂类型的对象、具有面向对象的继承、多态等特性。其最大的性能优势是不必像关系数据库一样在数据使用之前先对数据连接，而是以使用数据的方式存储数据，这就大大提高了性能。当

前的主要产品有 Versant、Objectivity、ObjectStore、DB4Objects。但是因为对象数据库没有提出一个高效的从关系模型到对象模型的转换，所以对象数据库系统使用的范围比较小，主要是在小型的嵌入式系统中，例如作为手机的数据存储模型。

（2）内存数据库

它打破了传统磁盘数据库的设计观念，考虑内存直接快速存取的特点，以 CPU 和内存空间的高效利用为目标来重新设计开发各种策略与算法、技术、方法以及机制。其本质特征是数据的“主拷贝”或“工作版本”常驻内存，活动事务只与内存数据库的拷贝打交道。当前主要的产品包括：TimesTen、SqlLite、eXtreme DB、Solid。但是因为其固有的处理策略，要求要有足够大的内存才能够实现，而内存空间是有限的，所以当前内存数据库主要用在实时业务系统中，例如在证券交易系统中，可以考虑使用内存数据库。

（3）分布式数据库

由通过网络相互连接、多个不同场地上的局部数据库构成。这些数据库服从于同一个应用需求，它们在业务逻辑上相互关联，由一个全局统一的分布式数据库管理系统进行管理。这种数据库类型具有可用性强、可扩展性强等特点。但是缺点是因为分布式的引入，使数据库管理系统的复杂性大大增加，对分布式控制提出了更高的要求和挑战。所以目前没有很成熟的真正意义上的分布式数据库产品，虽然有些关系数据库系统通过扩充实现的分布式存储，但是没有像分布式数据库描述的那样，实现事务级别的分布式。

（4）并行数据库

它是在并行机上运行的具有并行处理能力的数据库系统。并行数据库系统是数据库技术与并行计算技术相结合的产物。并行计算技术利用多处理机并行处理产生的规模效益来提高系统的整体性能，为数据库系统提供了一个良好的硬件平台。比较有代表性的并行数据库包括 Vertica、Aster Data 的 nCluster、NCR 的 Teradata 以及 Greenplum。

2. 数据建模方法

（1）Richard Barker 表示法

Richard Barker 表示法最初是由英国的 CACI 咨询公司提出的，现在是欧洲所采用的 SSADM 方法的一部分。后来得到了 Richard Barker 的大力倡导，为 Oracle 公司采用，成为其“CASE* 方法”的一部分。

在需求分析中宜采用 Barker 数据建模表示法，这是因为它外观简洁，最易于向用户进行表述，对于非技术人员来说，清晰的模型图更容易理解。在具有同等完整性的表示法中，Barker 表示法是最简单、最清楚的一种。它使用的符号种类相对较少，表示出的模型图比较清楚，要求用户认知的成分种类也比较少。

在实体的类型与属性的表示上，其他一些表示法额外地添加了一些符号，如 IDEF1X 表示法中，用不同的符号来表示“依赖”和“独立”实体类型，同时还需要在不同的实体类型端使用不同的关系符号。UML 表示法中根据参照完整性的不同而用两种特殊的符号来指明“属于”和“包

括”的关系。这些所添加的符号都增加了模型图的复杂性，使其更难以理解，而对于用户而言，却并没有增加通过 Barker 表示法的简单符号和名称就能表示的信息之外的有效内容。

在关系的表示上，对于大多数用户来说，可选择性是最重要的特性，在 Barker 表示法中，可选择性是通过最明显的外观特点—关系线的虚实来表示的。IDEF1X 表示法中虽然也利用了关系线的虚实，但却是用来表达某个关系在多大程度上是唯一标识。对于关系的基数性，Barker 表示法只需要用乌鸦脚符号的有无就完全可以表示出某个关系的上限，而在其他一些表示法中，表示手段就要复杂得多。例如 IDEF1X 表示法中通过虚实线与其两端不同的圆形、菱形符号的 22 种组合分别表示不同的基数性和可选择性关系；UML 表示法中也需要通过详细定义上下限来表示。

在关系名称的表示上，Barker 表示法要求分析师对各种关系做出精炼的表述，而且要使用清楚、语法正确、易于理解的介词或介词短语，这方面其他表示法都是采用动词或动词短语作为关系名称的。由于在自然语言中，介词就是用以描述关系的，所以 Barker 表示法更为合适。动词描述的是动作而不是关系，用动词来描述关系其实是对两个实体类型间的动作来下定义，在数据模型中简单地描述关系本身更为合适，而动词则更适用于功能模型中。

在子类型的表示上，Barker 表示法是唯一在超类型内部表示子类型的表示法，这样做一方面可以强调子类型是超类型的子集这一事实，另一方面也可以节省绘图空间。在模型的可读性方面，其他表示法中都允许关系线任意弯折，从而使模型图显得更为复杂。而在 Barker 表示法中对模型图的布局有着特定的要求，可以使关系线尽可能短、直。

（2）IDEF1X 表示法

IDEF1X 表示法是为美国联邦政府的众多部门所采用的一种数据建模技术。在 IDEF 方法家族中，IDEF1 是最早着手研究和制定的标准之一，它主要用作开发“信息模型”。而 IDEF1X 标准则是在 IDEF1 标准的基础上进行的扩展，它在表示符号和语义方面进行了较大的改动，主要增加了概括和聚合语义，于 1985 年正式发布。1993 年 IDEF1X 成为美国政府的处理标准文件 FIPS（Federal Information Processing Specification）之一。

在关系数据库设计中宜采用 IDEF1X 表示法，这是因为对于技术人员而言，IDEF1X 表示法能表示出实现方式上的复杂细节，例如突出强调的外键存在、关系的可选择性和基数性的完整组合等。但对非技术人员而言，这种方法并没有遵从设计优良图形的原则，该方法可能较难掌握和使用。在关系的表述上，IDEF1X 表示法中所用的符号并不与它们所要表达的模型中的概念完全相符。在这种表示法中，本应由一个符号表示的概念却要求同时使用多个符号来表示，而且某一特定情况可以由不只一组符号来表示，而同一符号在不同上下文中又可能表示不同的含义。某一特定情况到底该用哪个符号来表示在很大程度上取决于上下文和该关系的实现方式，而不是取决于该情况本身。这些做法导致要教会非技术人员读懂 IDEF1X 模型图是极为困难的。

IDEF1X 表示法中，直线的虚实用来表示某一实体类型是否构成另一实体类型的唯一标识（主键）。这将要求分析师在开始考察模型中各关系的可选择性或基数性之前，先分析依赖性。但在实际建模时，分析师通常总是先考察哪些实体类型是其他实体类型所需要的，以及其中涉及多少次出现等问题，通常是在后期才考虑主键或标识等细节。另外对这种模型的修改也是非常困难的，如果基数性或可选择性方面出现了某个小错误，就必须改动好几个符号。

（3）UML 表示法

面向对象是软件领域的主流技术，UML 的诞生与发展更是对于整个软件开发流程的改善起了相当关键的作用。20 世纪 80 年代，出现了一大批面向对象的分析与设计方法，如 Booch 方法、Coad / Yourdon 方法、Firesmith 方法、Jacobson 的 OOSE、Martin / Odell 方法、Rumbaugh 等人的 OMT、Shlaer / MelIor 方法等，设计与建模技术比较纷乱，众说纷纭。1997 年 Rational 公司与其他公司一起共同推出了 UML 1.0，形成了统一的建模语言，2003 年又推出了 UML2.0，2.0 可以让开发者在设计软件的过程中分解成不同层次进行设计，同时 2.0 完整度非常高，弥补了 1.0 版在做软件设计开发方面的缺陷。

在面向对象的设计中宜采用 UML 表示法，这是因为它更完整、更详尽。通过其他数据建模方法所不具备的额外表达能力，使其不仅适合于系统的逻辑分析模型的表述，也同样适用于物理设计模型的表述。

在实体属性的表述上，UML 表示法较其他表示法能够更详细地对属性做出描述。可以标注出包括构造型、可视性、名称、多样性、类型、初始值等一个或多个内容，而在 Barker 及 IDEF1X 中对于实体属性仅标注出名称。

在关系的表述上，对并非是两个关联间的简单关系的商业规则，UML 表示法引入了一种小标志，该标志中可以包括描述任何商业规则的文字。对关系的可选择性和基数性的表示，UML 表示法能够表述更为复杂的上限，可以具体说明某个实体类型的出现可能与另一个实体类型的 1、7～9 或 10 次出现有关。

在关系间约束的表述上，UML 表示法用两个关联之间的简单直线代替了 Barker 表示法中对关系之间约束的表述，而且可以在这种直线上加评注，从而描述两个关联之间的任何关系。

5.3 数据查询、分析与建模技术的实现与工具

本部分内容主要对数据查询、分析与建模工具进行介绍，并给出了多种实现的技术。数据查询部分主要针对 Nutch 与 Lucene 进行介绍；数据分析部分主要对 Google 的 Dremel 技术进行深入分析；在数据建模部分，主要对 Sybase Power Designer 以及 ERWin 进行介绍，使得读者对数据查询、分析与建模有一个表全面的了解。

5.3.1 数据查询相关技术实现与工具

随着云计算技术以及大数据概念的提出，众多针对海量数据查询处理的技术开始出现，其中最为著名的就属于 Apache 的开源项目：Lucene 与 Nutch。

1. Lucene

（1）Lucene 介绍

Lucene 是 Apache 软件基金会 Jakarta 项目组的一个子项目，是一个用 Java 编写的开放源代码的

全文检索引擎工具包，即它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎及部分文本分析引擎。Lucene 可以对任何数据做索引和搜索。不管数据源是什么格式，只要它能被转化为文字的形式，就可以被 Lucene 所分析利用。也就是说不管是 Word、HTML、PDF 还是其他形式的文件，只要可以从中抽取出文字形式的内容就可以被 Lucene 所用，就可以用 Lucene 对它们进行索引及搜索。Lucene 由 Apache 软件基金会支持和提供。原作者为 Doug Cutting，他是一位资深的全文索引/检索专家。如图 5.4 所示为 Lucene 的体系结构图。

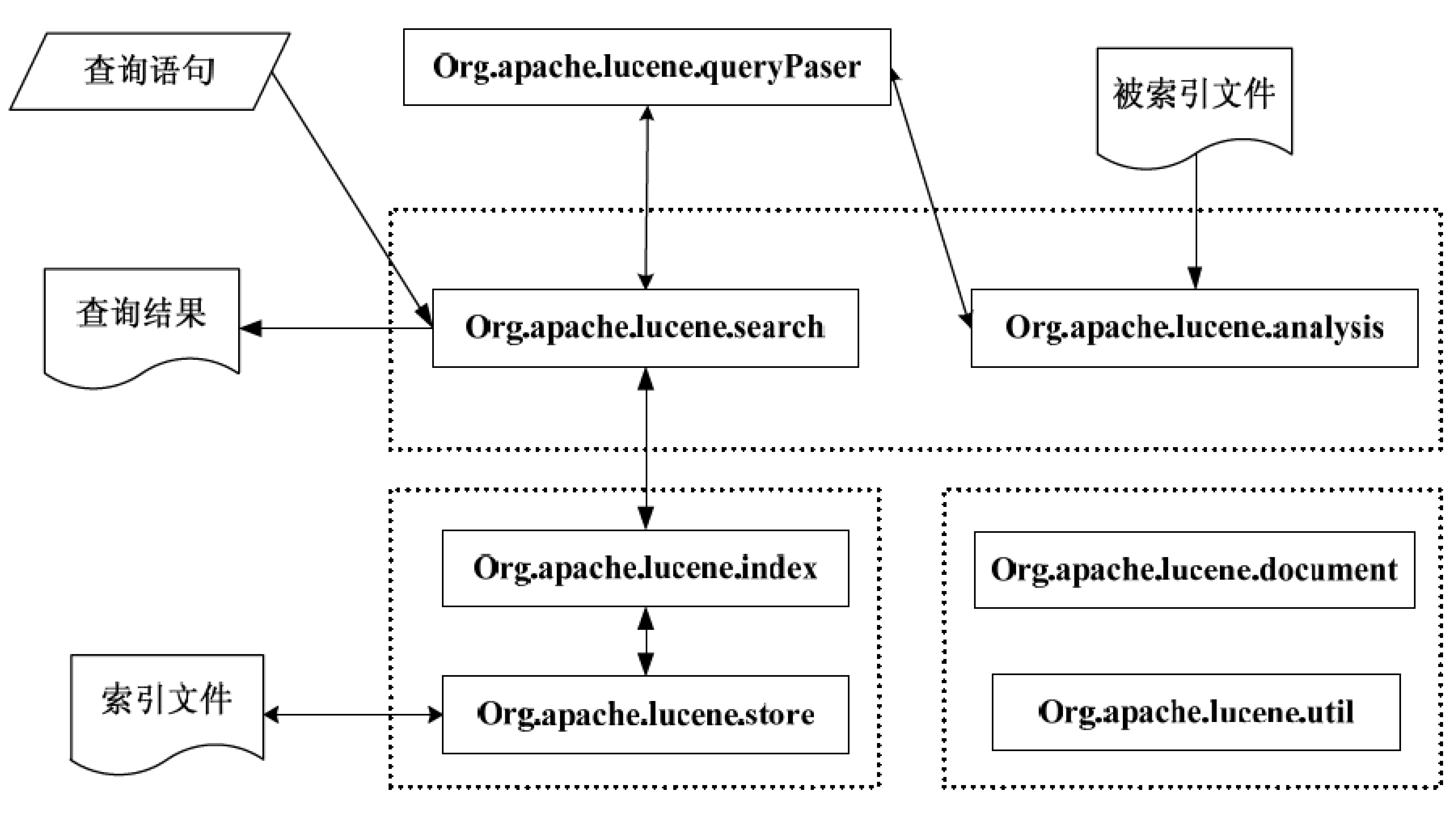


图 5.4　Lucene 体系结构图

Lucene 源码中共包括 7 个子包，每个包完成特定的功能。如表 5.1 所示为 Lucene 源码包及其对应功能。

表 5.1　Lucene 源码包及其对应功能

包名	功能
Org.apache.lucene.analysis	语言分析器，主要用于切分词
Org.apache.lucene.document	索引存储时的文档结构管理
Org.apache.lucene.index	索引管理，包括索引的建立、删除等
Org.apache.lucene.queryParser	查询分析器，实现查询关键词间的运算
Org.apache.lucene.search	检索管理，根据查询条件，检索得到结果
Org.apache.lucene.store	数据存储管理
Org.apache.lucene.util	公共类

从根本上说，主要包括两块：一是文本内容经切分词后索引入库；二是根据查询条件返回结果，即索引部分和查询部分两部分。要看源码包所对应完成的功能属于索引部分还是查询部分。

（2）Lucene 的索引

Lucene 索引信息存储有三种可选方式：内存（RAM）、文件系统（FS）和数据库（DB）。RAM 存储适用于较小的检索系统，文件系统存储可用于中型检索，数据库存储则适用于对检索性能要

求更高的系统。下面对其索引存储逻辑进行分析。在 Lucene 中，索引（index）由段（segment）组成，段（segment）由记录（document）组成，记录（document）由域（field）组成，域（field）由字符串（term）组成。例如，一个 document 可以与一个物理文件对应，将其中文件名、文件内容、文件创建时间等信息提取为数据源放入 document 中，也可将多个物理文件的数据源同时放入一个 document。数据源是由一个被称为 field 的类表示。这个 field 类主要用来标识当前数据源各种属性，主要是以下三种标识。

- 可分词性：该数据源是否需要经过分词。可分词时，内容被分成多个可被索引的词；不可分词时，整个字段内容作为一个词。
- 可存储性：字段内容直接按照词存放，而不是以倒排形式存放。
- 可索引性：该数据源的数据是否要在用户检索时被检索，需要检索时，字段内容以倒排形式存放，即记录字段每个词在某一文档中出现的频率。

从代码方面上看，在 Lucene 的索引部分。invertDocument()方法是最重要的，由它调用分析器的接口，分析统计词条的位置与频率信息。作用就是建立倒排索引，直观地讲，就像一本书在最后给出书中出现的名词列表，同时对应给出该名词出现在第几章和第几页。在查找的时候，可以直接定位到具体页码，而不用从目录开始逐个查找。Lucene 在维护和扩展索引的时候不断创建新的文件，最终将这些新的小索引文件并入大索引中。程序员可以根据不同的要求自行调整批次大小、周期长短。这点对于 Lucene 的检索效率也相当的重要。建立索引，是需要占用内存资源的，当有新的记录加入索引时，并不直接写入硬盘而是先放在内存中，所以最直接提高检索速度的方法就是提高内存存放索引的缓冲区的大小。具体缓冲区的大小设置需要根据实际情况而定。如图 5.5 所示为 Lucene 全文索引及检索过程的描述。

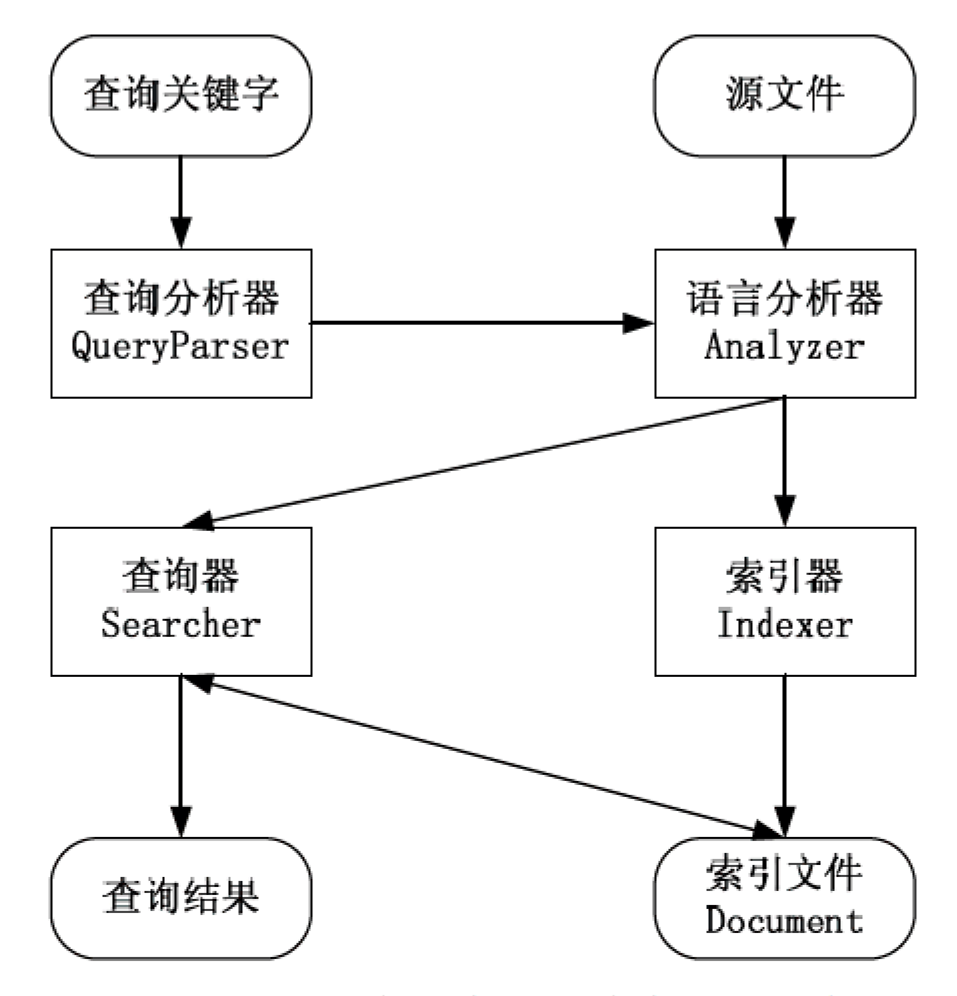


图 5.5 Lucene 全文索引及检索过程示意图

（3）Lucene 和其他全文检索的区别

传统的查找技术是通过逐次匹配内存中的文本实现的，即顺序查找。对文档集合中的信息进行少量预处理或不做处理，这种方法只适合文档较少的情况，虽然结构简单，易实现，但检索速

度比较慢，尤其在处理海量数据和模糊查询时有着明显不足。当信息量在 TB 级别时，查找的速度是无法忍受的。Lucene 通过特殊的索引结构实现了传统检索不擅长的全文索引机制，这也是 Lucene 快速发展的原因之一。表 5.2 所示为对 Lucene 和其他全文检索的比较。

表 5.2 Lucene 与其他全文检索的比较

	Lucene	其他开源全文检索系统
增量索引和批量索引	可以进行增量的索引（Append），可以对于大量数据进行批量索引，并且接口设计用于优化批量索引和小批量的增量索引	很多系统只支持批量的索引，有时数据源有一点增加也需要重建索引
数据源	Lucene 没有定义具体的数据源，而是一个文档的结构，因此可以非常灵活地适应各种应用（只要前端有合适的转换器把数据源转换成相应结构）	很多系统只针对网页，缺乏其他格式文档的灵活性
索引内容抓取	Lucene 的文档是由多个字段组成的，甚至可以控制哪些字段需要进行索引，哪些字段不需要索引，近一步索引的字段也分为需要分词和不需要分词的类型： 需要进行分词的索引，比如标题、文章内容字段 不需要进行分词的索引，比如作者/日期字段	缺乏通用性，索引整个文档
语言分析	通过语言分析器的不同扩展实现： 可以过滤掉不需要的词：an the of 等， 西文语法分析：将 jumps jumped jumper 都归结成 jump 进行索引/检索 非英文支持：对亚洲语言，阿拉伯语言的索引支持	缺乏通用接口实现
查询分析	通过查询分析接口的实现，可以定制自己的查询语法规则，比如多个关键词之间的“+”、“-”、“and”和“or”关系等	
并发访问	能够支持多用户的使用	

表 5.3 所示为 Lucene 全文索引与数据库模糊查询的比较。

表 5.3 Lucene 全文索引与数据库模糊查询的比较

	Lucene 全文索引引擎	数据库
索引	将数据源中的数据都通过全文索引——建立反向索引	对于 LIKE 查询来说，数据传统的索引是根本用不上的。数据需要逐个便利记录进行 GREP 式的模糊匹配，比有索引的搜索速度要有多个数量级的下降
匹配效果	通过词元（term）进行匹配，通过语言分析接口的实现，可以实现对中文等非英语的支持	使用 like "%net%" 会把 netherlands 也匹配出来。多个关键词的模糊匹配：使用 like "%com%net%"就不能匹配词序颠倒的 xxx.net..xxx.com
匹配度	有匹配度算法，将匹配程度（相似度）比较高的结果排在前面	没有匹配程度的控制：比如有记录中 net 出现 5 词和出现 1 次的，结果是一样的
结果输出	通过特别的算法，将匹配度最高的头 100 条结果输出，结果集是缓冲式的小批量读取的	返回所有的结果集，在匹配条目非常多的时候（比如上万条）需要大量的内存存放这些临时结果集

（续表）

	Lucene 全文索引引擎	数据库
可定制性	通过不同的语言分析接口实现，可以方便地定制出符合应用需要的索引规则（包括对中文的支持）	没有接口或接口复杂，无法定制
结论	高负载的模糊查询应用，需要负责的模糊查询的规则，索引的资料量比较大	使用率低，模糊匹配规则简单或者需要模糊查询的资料量少

大部分的搜索（数据库）引擎都是用 B 树结构来维护索引，索引的更新会导致大量的 IO 操作，Lucene 在实现中，对此稍微有所改进：不是维护一个索引文件，而是在扩展索引的时候不断创建新的索引文件，然后定期把这些新的小索引文件合并到原先的大索引中（针对不同的更新策略，批次的大小可以调整），这样在不影响检索的效率的前提下，提高了索引的效率。

2. Nutch

Nutch 是一个用 Java 语言实现的开放源代码的 Web 搜索引擎，是以 Lucene 为基础实现的搜索引擎应用程序，Lucene 为 Nutch 提供了文本索引和查询服务的 API，而 Nutch 在 Lucene 的基础上实现了网页收集，因此 Nutch 在总体架构上分为网页收集、建立索引和查询服务三个部分。

- 网页收集程序通过定期收集和增量收集方式从互联网中抓取网页，并将原始网页建立索引存入数据库中。
- 建立索引程序则从抓取过来的网页提取其中的 URL、标题、内容等关键词，将不同格式的数据源转换成其内部可以识别的文件格式，然后建立倒排文件，即用文档中的关键词作为索引，文档作为索引目标的一种结构，从而建立并维护索引库。
- 查询服务程序接收用户提交的查询词条，加以分词与过滤，在索引库及数据库中搜索相应的网页，并按照其内部评分算法对结果进行排序，返回结果。

如图 5.6 所示为 Nutch 结构图。

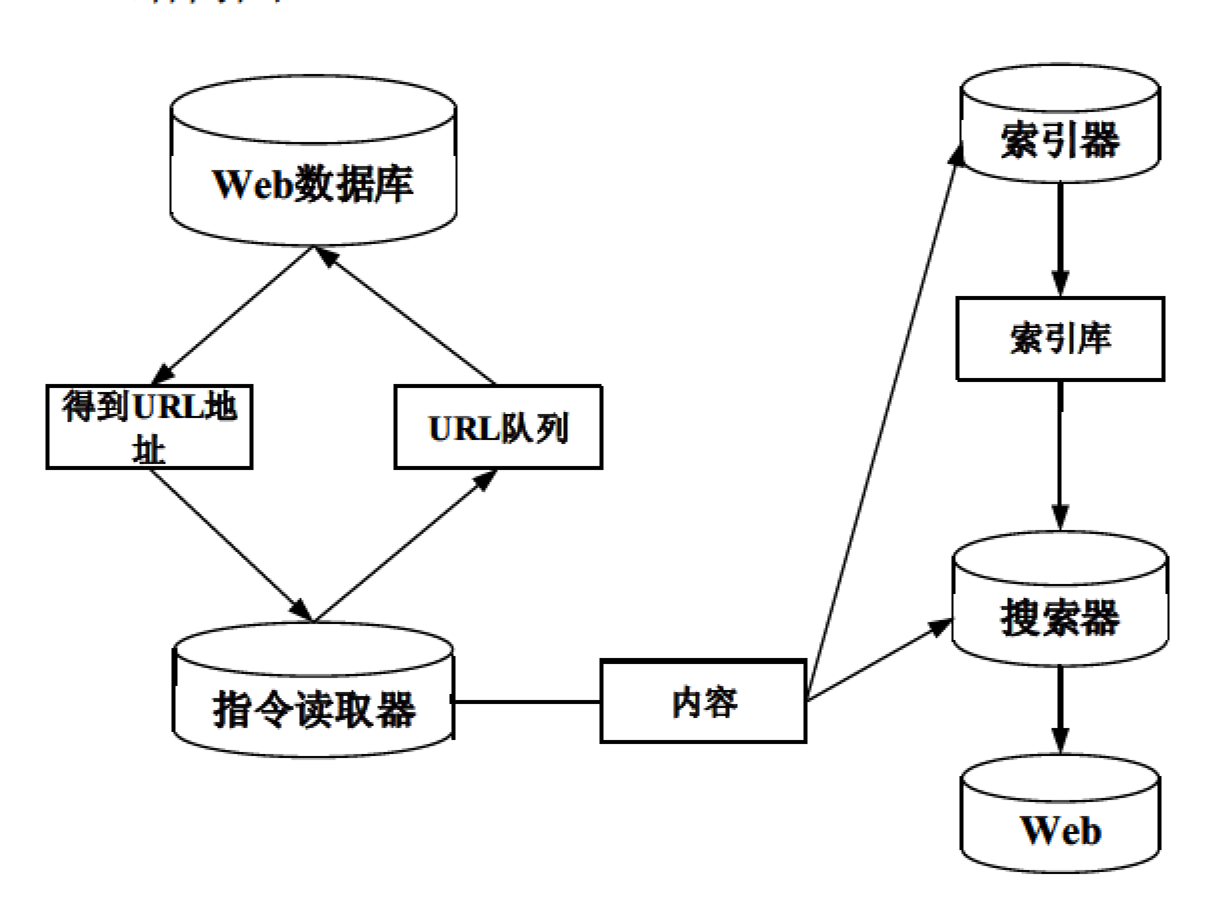


图 5.6　Nutch 结构图

由于 Nutch 没有商业目的，对学术和政府类站点的搜索来说，有着商业搜索引擎无法比拟的

优势。因为它能给用户显示一个公平的排序结果。Nutch 开源的特点，吸引了许多研究者，对它的排序算法也进行着不断的研究。因为 Nutch 极易扩展，研究者可以随意复制和修改源程序，同时也可以将其集成到用户应用程序中去，为用户提供更好的查询服务。概括来说，Nutch 搜索引擎特点总结如下。

- 高透明度，Nutch 属于开源程序，所以任何单位或个人都可以查看分布式搜索引擎的工作原理和工作过程。
- 可扩展性，Nutch 程序设置灵活，可以根据用户需求进行定制。
- 高稳定性，Nutch 通过长时间实际应用，结果表明运行非常稳定。

如图 5.7 所示为 Nutch 的工作流程

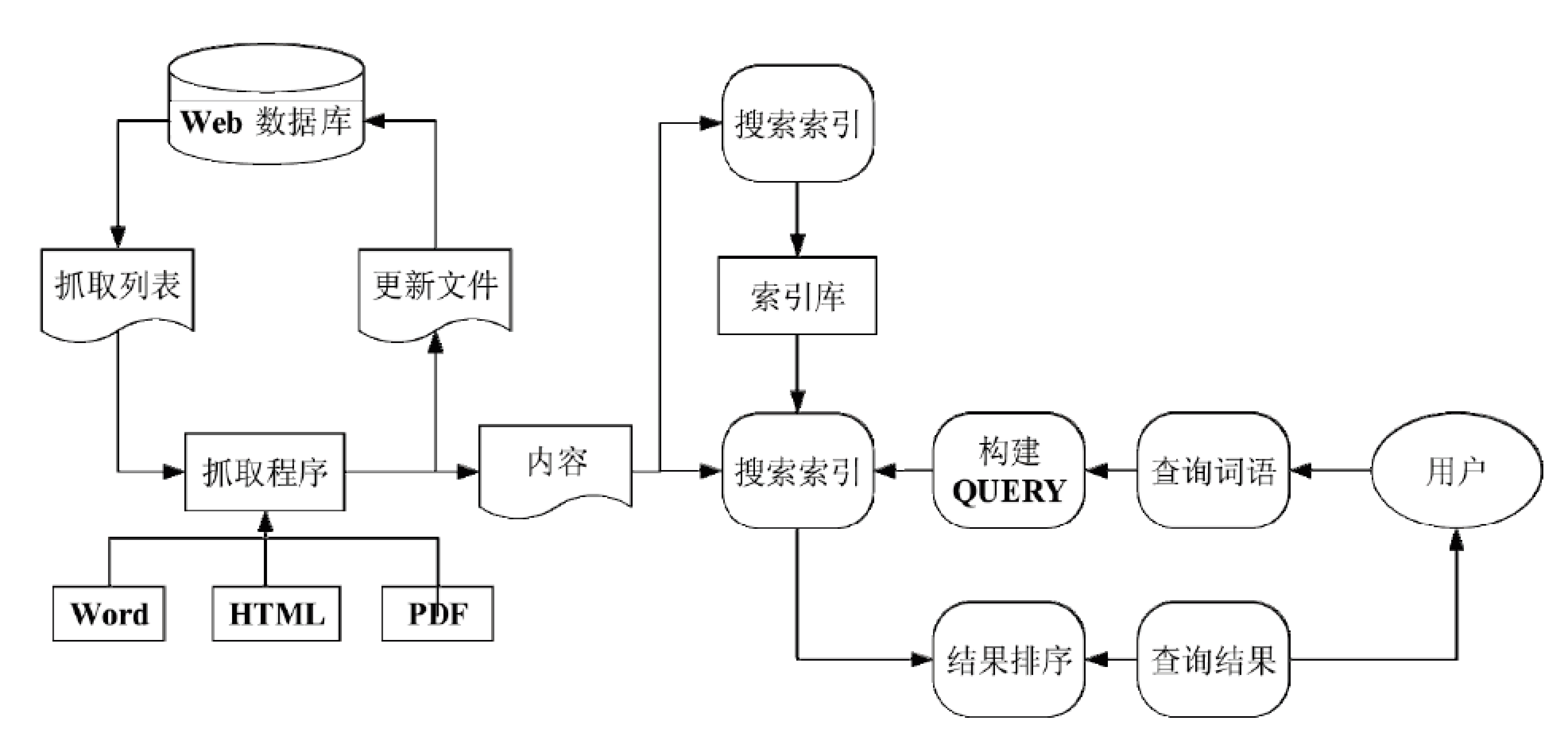


图 5.7　Nutch 的工作流程

3. Solr

作为 Apache 下基于 Java 的文本搜索引擎库 Lucene 的一个子项目，Solr 采用 Java5 开发，通过对 Lucene API 的扩展，形成了一个独立、高效率、高并发的企业级搜索应用服务器。Solr 提供了比 Lucene 更为丰富的查询语言和功能，同时实现了可配置、可扩展，并对查询性能进行了优化，有完善的功能管理界面，易于加入到 Web 应用程序中。Solr 支持多种输出格式（包括 XML/XSLT 和 JSON 格式），用户通过 HTTP 方式与 Solr 交互，HTTP GET 操作提出查找请求，并得到 XML 格式的返回结果，提交 XML 格式的文档，而由 Solr 生成索引文件。Solr 具备如下特点。

- 基于标准的开放接口：Solr 搜索服务器支持通过 XML、JSON、HTTP 查询和获取结果。
- 易管理与配置：Solr 可以通过 Web 页面管理，数据以 XML 输出，Solr 配置通过 XML 完成。
- 高效率分组（Facets）：高效率搜索结果自动分类，用户可以按照类型、时间等进行分组。
- 高亮功能：对匹配的字符自动在搜索结果中高亮显示。
- 可伸缩性：快速增量更新和快照分发/复制到其他服务器，支持多种方式的缓存。
- 灵活的插件体系：新功能能够以插件的形式方便地添加到 Solr 服务器上。
- 分布式搜索：支持单台或者多台服务器索引、搜索，大的索引可以分割成多个小部分，通

过 Solr 一次搜索多台服务器，通过分布式搜索来提高效率。

- 数据导入工具：数据库和其他结构化数据源都可以导入、映射和转化为 Lucene 的索引格式。

5.3.2 数据分析相关技术实现与工具

海量数据处理技术已经得到一定的发展，世界上各大 IT 公司争相推出自己的大数据分析工具，其中包括 Goolge、SAS 等。

1. SAS

（1）SAS 介绍

SAS（Statistical Analysis System）即统计分析系统，于 1966 年由美国 North Carolina 州立大学开始研制。被誉为数据处理和统计分析领域的国际标准软件系统，类似于 MATLAB 工具箱，其具有多个模块。

- 基本模块：Base SAS
- 统计分析模块：SAS / STAT
- 高级绘图模块：SAS / GRAPH
- 矩阵运算模块：SAS / IML
- 运筹学和线性规划模块：SAS / OR
- 经济预测和时间序列分析模块：SAS / ETS 等

（2）SAS 企业级智能平台的主要特色

- 一致：能够在整个企业内对元数据进行整合、共享和集中管理，全面了解企业的真实情况，消除信息竖井以及相关的信息维护成本。
- 开放：与各种数据和元数据源整合，跨越各种孤立的 IT 环境为信息流提供支持。
- 统一：跨越组件和应用共享通用的安全和管理服务、数据存储和管理功能、查询和报表工具、分析工具包、出版基础架构以及基于 Web 的用户界面。
- 可管理：通过单一管理点对 SAS 企业级智能平台的所有要素进行集中管理。
- 基于标准：支持行业标准协议、编程语言、模型和通信接口。
- 可扩展：能够让客户通过标准开发环境（例如 Java）和完善的应用编程接口（API）开发定制的解决方案。
- 可移动：能够根据不断变化的增长和维护需求将应用在系统和平台之间进行移动。
- 可扩展：企业级的网格计算功能以最有效的方式对所有计算资源进行利用。
- 兼容：与企业的 IT 实践、技术、方法和基础架构兼容，充分利用客户现有的技术投资。
- 能够满足未来需求：可扩展性可以满足未来增长需求；可定制性能够支持新的特殊应用；开放性和基于标准能够支持尚未定义的未来技术。

（3）高性能分布式分析

高性能分析环境是用户可以充分利用 IT 投资，同时克服原有架构的约束，从大数据资产中产生高价值的洞察。 以下列出了三种分布式技术，合理地组合利用这些技术，可以满足不断发展的业务需求。

- SAS In-Memory Analytics（内存分析）

大数据和复杂的分析运算都在内存中进行处理，并分布到多个专用节点上来执行，从而生成高度准确的洞察，以近实时的方式来解决复杂问题。

- SAS In-Database（库内分析）

数据整合与分析功能都是在数据库内执行的，无需数据迁移和转换，就可以更好地进行数据治理，加快获得洞察的速度。

- SAS Grid Computing（网格计算）

SAS 作业是在共享的集中管理的 IT 资源池中处理，在提高效率的同时，也降低了成本，带来更高的性能。

SAS 采用这些高性能技术来解决大量的业务问题，展开各种分析，包括数据可视化、数据探索以及模型的开发与部署。借助 SAS 高性能分析的灵活性和扩展能力，不管数据量有多大，也不管数据和分析方法有多复杂，SAS 都能很好地完成任务。

2. Google Dremel

BigQuery 是真正为大数据而生的企业级云计算产品，其核心是云平台的一项基础服务（PaaS），用于对 TB 级别的大数据进行实时的分析处理。单纯从技术上来看，BigQuery 就是一个在云端的 SQL 服务（类 SQL），提供对海量数据的实时分析；BigQuery 查询基于 Dremel 技术。

Dremel 是一个用于分析只读嵌套数据的可伸缩、交互式 ad-hoc 查询系统。通过结合多层级树状执行过程和列状数据结构，它能做到几秒内完成万亿行数据之上的聚合查询。此系统可伸缩至成千上万的 CPU 和 PB 级别的数据，而且在 Google 已有几千用户。下面将对 Dremel 进行详细介绍。

（1）简介

大规模分析型数据处理在互联网公司乃至整个行业中应用已经越来越广泛，尤其是因为目前已经可以用廉价的存储来收集和保存海量的关键业务数据。如何让分析师和工程师便捷地利用这些数据也变得越来越重要；在数据探测、监控、在线用户支持、快速原型、数据管道调试以及其他任务中，交互的响应时间一般都会造成本质的区别。

执行大规模交互式数据分析对并行计算能力要求很高。例如，如果使用普通的硬盘，希望在 1 秒内读取 1TB 压缩的数据，则需要成千上万块硬盘。相似的，CPU 密集的查询操作也需要运行在成千上万个核心上。在 Google，大量的并行计算是使用普通 PC 组成的共享集群完成的。一个集群通常会部署大量共享资源的分布式应用，各自产生不同的负载，运行在不同硬件配置的机器

上。一个分布式应用的单个工作任务可能会比其他任务花费更多的时间，或者可能由于故障或者被集群管理系统取代而永远不能完成。因此，处理好异常、故障是实现快速执行和容错的重要因素。

互联网和科学计算中的数据经常是独立的、互相没有关联的。因此，在这些领域一个灵活的数据模型是十分必要的。在编程语言中使用的数据结构、分布式系统之间交换的消息、结构化文档等，都可以用嵌套式表达法来很自然地描述。规格化、重新组合这些互联网规模的数据通常是代价昂贵的。嵌套数据模型成为了大部分结构化数据在 Google 处理的基础。

Google 支持在普通 PC 组成的共享集群上对超大规模的数据集合执行交互式查询。不像传统的数据库，它能够操作原位嵌套数据。原位意味着在适当的位置访问数据的能力。比如，在一个分布式文件系统（比如 GFS）或者其他存储层（比如 Bigtable）。查询这些数据一般需要一系列的 MapReduce（MR）任务，而 Dremel 可以同时执行很多，而且执行时间比 MR 小得多。Dremel 不是为了成为 MR 的替代品，而是经常与它协同使用来分析 MR 管道的输出或者创建大规模计算的原型系统。

Dremel 从 2006 起投入使用并在 Google 有几千用户。多种多样的 Dremel 实例被部署在公司里，排列着成千上万个节点。使用此系统的地方包括：

- 分析网络文档
- 追踪 Android 市场应用程序的安装数据
- Google 产品的崩溃报告分析
- Google Books 的 OCR 结果
- 垃圾邮件分析
- Google Maps 的地图部件调试
- 托管 Bigtable 实例中 Tablet 的迁移
- Google 分布式构建系统中的测试结果分析
- 大量硬盘的磁盘 IO 统计信息
- Google 数据中心上运行的任务的资源监控
- Google 代码库的符号和依赖关系分析

Dremel 基于互联网搜索和并行 DBMS 的概念。首先，它的架构借鉴了用在分布式搜索引擎中的服务树概念。如一个 Web 搜索请求，查询请求被推入此树、在每个步骤被重写。通过聚合从下层树节点中收到的回复，不断装配查询的最终结果。其次，Dremel 提供了一个高级、类 SQL 的语言来表达 ad-hoc 查询。与 Pig 和 Hive 对照，它使用自己的技术执行查询，而不是转换为 MR 任务。

最关键的是，Dremel 使用了一个 column-striped 的存储结构，使得它能够从二级存储中读取较少数据并且通过更廉价的压缩减少 CPU 消耗。列存储曾被采用来分析关系型数据，但是还没有推广到嵌套数据模型上。这里的列状存储格式在 Google 已经有很多数据处理工具支持，包括 MR、Sawzall 以及 FlumeJava。

（2）实例背景

假设一个场景，来说明交互式查询处理的必要性，以及它在数据管理生态系统上怎么定位。假设某公司的员工 Wendy，想到一个新奇的点子，她想从 Web 网页中提取新类型的 Signals。为此运行一个 MR 任务，分析输入数据然后产生这种 Signals 的数据集合，在分布式文件系统上存储这数十亿条记录。为了分析她实验的结果，启动 Dremel 然后执行几个交互式命令：

```
DEFINE TABLE t AS /path/to/data/*
SELECT TOP(signal1, 100), COUNT(*) FROM t
```

她的命令只需几秒钟就执行完毕。她也运行了几个其他的查询来证实她的算法是正确的。她发现 Signals 中有非预期的情况，于是编写了一个 FlumeJava 程序执行更加复杂的分析式计算。一旦这个问题解决了，就建立一个管道，持续地处理输入数据。然后她编写了一些 SQL 查询来跨维度地聚合管道的输出结果，将它们添加到一个交互式的 Dashboard，其他工程师能非常快速地定位和查询结果。

上述实例要求在查询处理器和其他数据管理工具之间互相操作。第一个组成部分是一个通用存储层。Google File System（GFS）是 Google 公司中广泛使用的分布式存储层。GFS 使用冗余复制来保护数据不受硬盘故障影响，即使出现掉队者（Stragglers）也能达到快速响应时间。对原位数据管理来说，一个高性能的存储层是非常重要的。它允许访问数据时不消耗太多时间在加载阶段。这个要求也导致数据库在分析型数据处理中不经常使用。另外一个好处是，在文件系统中能使用标准工具便捷的操作数据，比如，迁移到另外的集群，改变访问权限，或者基于文件名定义一个数据子集。第二个构建互相协作的数据管理组件的要素，是一个共享的存储格式。列状存储已经证明了它适用于扁平的关系型数据，但是使它适用于 Google 则需要适配到一个嵌套数据模型。

（3）数据模型

本部分将介绍 Dremel 的数据模型以及其他一些术语。这种在分布式系统中经常面对的数据结构在 Google 中广泛使用，也提供了开源实现。这种类型是基于强类型嵌套记录的。它的抽象语法是：

$$\pi = dom \mid \langle A1 : \pi[*|?], \ldots, An : \pi[*|?] \rangle$$

π 是一个原子类型（一个 int、一个 string 等，比如 DocId）或者记录类型（指向一个子结构，比如 Name）。在 dom 中原子类型包含整型、浮点数、字符串等类型。记录则由一到多个字段组成。字段 i 在一个记录中命名为 Ai 以及一个标签（比如“?”或“*”，指明该字段是可选的或重复的）。重复字段（*）表示在一个记录中可能出现多次，是多个值的列表，字段出现的顺序是非常重要的。可选字段（?）可能在记录中不出现。如果不是重复字段也不是可选字段，则该字段在记录中必须有值，有且仅有一个。

实例 1

```
DocId: 10
Links
  Forward: 20
  Forward: 40
```

```
  Forward:  60
Name
  Language
    Code: 'en-us'
    Country:  'us'
  Language
    Code: 'en'
    Url : 'http://A'
Name
  Url : 'http://B'
Name
  Language
    Code: 'en-gb'
    Country:  'gb'
```

实例 2

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward: 80
Name
Url: 'http://C'
```

数据格式：

```
message Document {
       required int64 DocId;
       optional group Links {
             repeated int64 Backward;
             repeated int64 Forward; }
       repeated group Name {
             repeated group Language {
                   required string Code;
                   optional string Country; }
      optional string Url; }}
```

以上描述了一个叫 Document 的 Schema，表示一个网页。一个网页文档必有整型 DocId 和可选的 Links 属性，包含 Forward 和 Backword 列表，列表中每一项代表其他网页的 DocId。一个网页能有多个名字（Name），表示不同的 URL。名字包含一系列 Code 和（可选）Country 的组合（也

就是 Language）。上述的两个数据实例是基于固定模型的，模型的字段定义按照树状层级。一个嵌套字段的完整路径使用简单的点缀符号表示，如 Name.Language.Code。

嵌套数据模型为 Google 的序列化、结构化数据奠定了一个平台无关的可扩展机制。而且有为 C++、Java 等语言打造的代码生成工具。通过使用标准二进制 on-the-wire 结构，实现跨语言互操作性，字段值按它们在记录中出现的次序被顺序地排列。因此，一个 Java 编写的 MR 程序能利用一个 C++库暴露的数据源。如果记录被存储在一个列状结构中，快速装配就成为了 MR 和其他数据处理工具之间互操作性的重要因素。

如果要按照列式存储存储以上模型，总共要存储的有 6 列：

- Document.DocId
- Document.Links.Backward
- Document.Links.Forward
- Document.Name.Language.Code
- Document.Name.Language.Country
- Document.Name.Url

（4）列式存储

根据实例表格中的行式存储来看，可以将内部数据转换为列式存储——Dremel 中的实际存储格式。如下所示为上一节实例的列式存储格式。

（a）DocID

Value	R	D
10	0	0
20	0	0

（b）Name.Url

Value	R	D
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

（c）Links.Forward

Value	R	D
20	0	2
40	1	2
60	1	2
80	0	2

（d）Links.Backward

Value	R	D
NULL	0	1
10	0	2
30	1	2

（e）Name.Language.Code

Value	R	D
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

（f）Name.Language.Country

Value	R	D
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

如果是关系型数据，而不是嵌套的结构，存储的时候可以将每一列的值直接排列下来，不用引入其他的概念，也不会丢失数据。对于嵌套的结构，还需要两个变量 R（Repetition Level）和 D（Definition Level）才能完整无损地存储全部信息。

Dremel 的列式存储最大限度地分离了每个列，这是一种针对快速查询海量 Column-base DB 的设计方法。我们知道在 BigTable 和 HBase 中存储的多是海量稀疏数据，Column 的数量可能很大，但每次查询涉及的 column 经常很少。Dremel 的列存储模型最大限度地减少了数据访问量，只访问需要的数据。除“文件存储+MR”、“BigTable 存储+RowKey 查询+MR”之外，Dremel 又提供了多一种存储和查询方式。

（5）按行存储的数据转换为列存储

Dremel 使用的是新定义的列式存储数据，因此需要将已有的嵌套式数据（PB）转换成列式存储。Dremel 实际上是为每一列生成一个转换器（Writer），但由于列可能很多，而且是稀疏矩阵，为了避免不必要的浪费，Dremel 使用了一个转换树（Tree of Writer），只有用到的 Writer 才会被挂到树上。转换的伪代码如下：

```
procedure DissectRecord (RecordDecoder  decoder,
              FieldWriter  writer , int  repetitionLevel):
    Add current repetitionLevel and definition level to  writer
    seenFields = {} // empty set of integers
    while decoder has more field values
      FieldWriter  chWriter  =
        child of writer  for field read by  decoder
      int  chRepetitionLevel =  repetitionLevel
      if set seenFields contains field ID of  chWriter
       chRepetitionLevel =  tree depth of  chWriter
     else
        Add field ID of chWriter  to  seenFields
     end if
     if chWriter  corresponds to an atomic field
        Write value of current field read by decoder
              using  chWriter  at chRepetitionLevel
     else
        DissectRecord(new  RecordDecoder  for nested record
                  read by decoder, chWriter , chRepetitionLevel)
     end if
   end while
end procedure
```

（6）查询语言

Dremel 的查询语言（DSQL）主要是基于 SQL 修改的。DSQL 的主要特点是它是为处理嵌套式数据设计的，输入是一个或多个嵌套式数据表，输出不是数据集，也是一个嵌套式数据表。

DSQL 查询语句可以分为 3 个主要部分。

- 查询内容（FROM t）

表示只从表 t 中查询数据。

- 查询条件 （WHERE 语句）

可以用正则表达式、大于、小于、等于等。

- 生成结果

用原嵌套数据中的那些字段生成结果中的字段。

DSQL 非常强大，支持包括子查询、内联聚合查询、top-k、joins 等功能。

（7）Dremel 查询方式

由于 Dremel 使用上述的查询语句，而且数据是只读的，会密集地发起多次类似的请求。所以可以保留上次请求的信息，优化下次请求的 explain 过程。如图 5.8 所示为 Dremel 的查询过程及系统结构。

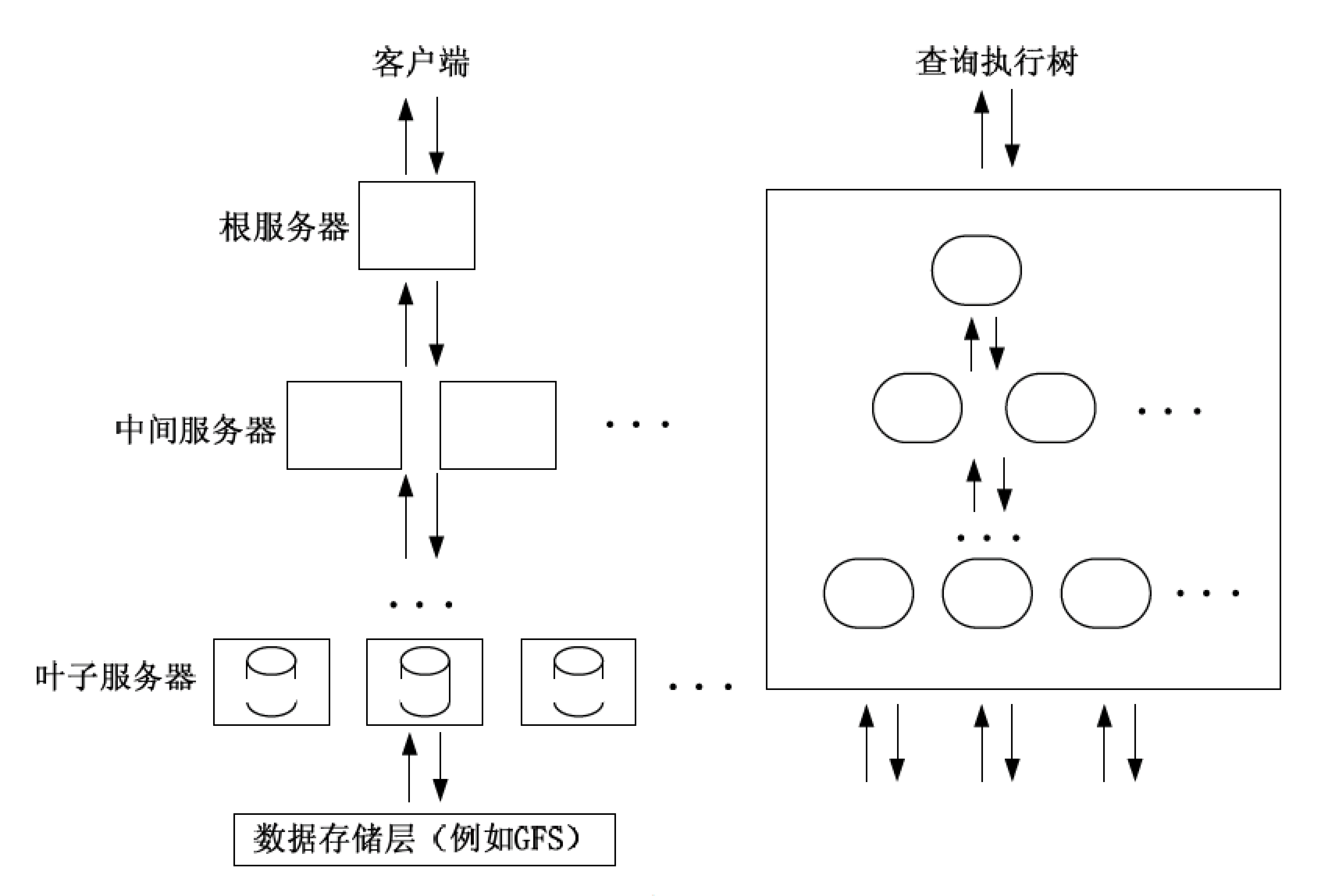


图 5.8　Dremel 系统架构以及内部节点执行图

图 5.8 是一个树状架构。当 Client 发起一个请求，根节点收到请求，根据 metadata，将其分解到枝叶，直到位于数据上面的叶子 Server。它们扫描处理数据，又不断汇总到根节点。对于请求：

```
SELECT A, COUNT(B) FROM T GROUP BY A
```

根节点收到请求，会根据数据的分区请求，将请求变成可以拆分的样子。原来的请求会变为：

```
SELECT A, SUM(c) FROM (R1 UNION ALL ... Rn) GROUP BY A
```

R1,…Rn 是 T 的分区计算出的结果集。越大的表有越多的分区，越多的分区可以越好地支持并发。

然后再将请求切分，发送到每个分区的叶子 Server 上面去，对于每个 Server：

```
Ri = SELECT A, COUNT(B) AS c FROM Ti GROUP BY A
```

结构集一定会比原始数据小很多，处理起来也更快。根服务器可以很快地将数据汇总。具体的聚合方式，可以使用现有的并行数据库技术。

Dremel 是一个多用户的系统。切割分配任务的时候，还需要考虑用户优先级和负载均衡。对于大型系统，还需要考虑容错，如果一个叶子 Server 出现故障或变慢，不能让整个查询也受到明显影响。

通常情况下，每个计算节点执行多个任务。例如，技巧中有 3000 个叶子 Server，每个 Server 使用 8 个线程，就可以有 24 000 个计算单元。如果一张表可以划分为 100 000 个区，就意味着大约每个计算单元需要计算 5 个区。在执行的过程中，如果某一个计算单元太忙，会另外启动一个来计算。这个过程是动态分配的。

对于 GFS 这样的存储，一份数据一般有 3 份拷贝，计算单元很容易就能分配到数据所在的节点上，典型的情况可以到达 95%的命中率。

Dremel 还有一个配置，就是在执行查询的时候，可以指定扫描部分分区，比如可以扫描 30% 的分区，在使用的时候，相当于随机抽样，加快查询。

（8）Dremel 实验

下面通过 Dremel 与 MapReduce 的实验对比来说明 Dremel 的优越性能。

表 5.4 所示为本次实验的数据相关信息。

表 5.4　实验数据信息

表名	记录数	大小（已压缩）	列数	数据中心	复制数量
T1	85 billion	87 TB	270	A	3×
T2	24 billion	13 TB	530	A	3×
T3	4 billion	70 TB	1200	A	3×
T4	1+ trillion	105 TB	50	B	2×
T5	1+ trillion	20 TB	30	B	3×

实验内容是常见的 CountWords 测试。MR 和 Dremel 都跑在 3000 个节点上。查询语句如下：

SELECT SUM(CountWords(txtField)) / COUNT(*) FROM T1

如图 5.9 所示为数据在不同状态下所花费的时间。

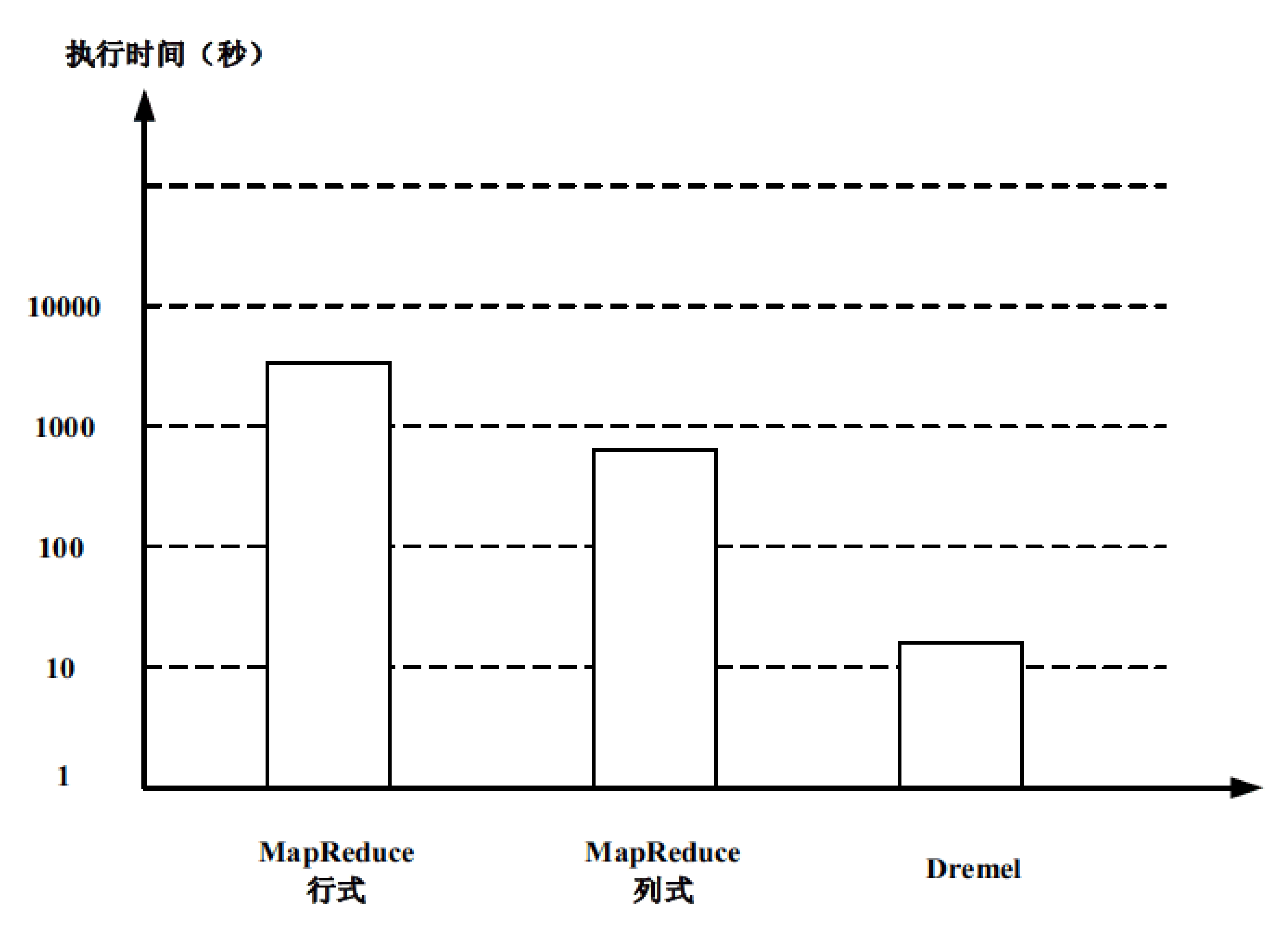


图 5.9　测试数据在不同状态下的执行时间

使用列式存储之后，访问的数据量从 85T 减少到了 0.5T，MR 的时间从几小时减少到了几分钟。Dremel 的查询树显然更加优越，将查询时间由几分钟缩短到了十几秒。

（9）Dremel 与 Hadoop

Dremel 相关论文中已经明确指出，Dremel 不是用来替代 MapReduce，而是和其更好的结合。Hadoop 的 Hive、Pig 无法提供及时的查询，而 Dremel 的快速查询技术可以给 Hadoop 提供有力的补充。同时 Dremel 可以用来分析 MapReduce 的结果集，只需要将 MapReduce 的 OutputFormat 修改为 Dremel 的格式，就可以在几乎不引入额外开销的情况下，将数据导入 Dremel。使用 Dremel 来开发数据分析模型，MapReduce 来执行数据分析模型。

（10）Dremel 的开源实现

Google Dremel 为 Google 内部使用，当前并未开源，但 Apache 已经推出 Dremel 的开源版本——Drill。Drill 有和 Dremel 相似的架构和能力。Drill 被寄希望于为 Hadoop 提供快速查询能力，并成为 Hadoop 上的重要组成部分。

现在 Drill 的目标是完成初始的需求和架构，完成一个初始的实现。这个实现包括一个执行引擎和 DrQL。目前，Drill 已经完成的需求和架构设计总共分为了 4 个组件。

- Query language:类似 Google BigQuery 的查询语言，支持嵌套模型，名为 DrQL。
- Low-lantency distribute execution engine:执行引擎，支持大规模扩展和容错。可以运行在上万台机器上计算数以 PB 的数据。
- Nested data format:嵌套数据模型，与 Dremel 类似。也支持 CSV、JSON、YAML 类似的模型。这样执行引擎就可以支持更多的数据类型。
- Scalable data source: 支持多种数据源，现阶段以 Hadoop 为数据源。

（11）Dremel 的特点

- Dremel 是一个大规模系统

Dremel 在一个 PB 级别的数据集上面，将任务缩短到秒级，无疑需要大量的并发。磁盘的顺序读速度在 100MB/s 上下，那么在 1s 内处理 1TB 数据，意味着至少需要有 1 万个磁盘的并发读。Google 一向善于利用廉价机建立性能强大的集群但是机器越多，出问题概率越大，如此大的集群规模，需要有足够的容错考虑，保证整个分析的速度不被集群中的个别慢（坏）节点影响。

- Dremel 是 MapReduce 交互式查询能力不足的补充

Dremel 和 MapReduce 一样，Dremel 也需要和数据运行在一起，将计算移动到数据上面。所以它需要 GFS 这样的文件系统作为存储层。在设计之初，Dremel 并非是 MapReduce 的替代品，它只是可以执行非常快的分析，在使用的时候，常常用它来处理 MapReduce 的结果集或者用来建立分析原型。

- Dremel 的数据模型是嵌套（Nested）的

互联网数据常常是非关系型的。Dremel 还需要有一个灵活的数据模型，这个数据模型至关重要。Dremel 支持一个嵌套的数据模型，类似于 Json。而传统的关系模型，由于不可避免地有大量的 Join 操作，在处理如此大规模的数据的时候，往往是有心无力的。

- Dremel 中的数据是用列式存储的

使用列式存储分析的时候，可以只扫描需要的那部分数据，减少 CPU 和磁盘的访问量。同时列式存储是压缩友好的，使用压缩，可以综合 CPU 和磁盘，发挥最大的效能。对于关系型数据，如果使用列式存储的技术比较成熟，但是对于嵌套（nested）的结构，Dremel 也可以用列存储。其中存储技术值得借鉴。

- Dremel 结合了 Web 搜索和并行 DBMS 技术

首先，Dremel 借鉴了 Web 搜索中的“查询树”的概念，将一个相对巨大复杂的查询，分割成较小较简单的查询。大事化小，小事化了，能并发地在大量节点上跑。其次，与并行 DBMS 类似，Dremel 可以提供一个 SQL-like 的接口。

5.3.3　数据建模相关技术实现与工具

1. Sybase PowerDesigner

（1）PowerDesigner 介绍

PowerDesigner 是 Sybase 公司的 CASE 工具集，使用它可以方便地对管理信息系统进行分析设计，它几乎包括了数据库模型设计的全过程。利用 PowerDesigner 可以制作数据流程图、概念数据模型、物理数据模型，可以生成多种客户端开发工具的应用程序，还可为数据仓库制作结构模型，也能对团队设备模型进行控制。 它可配合许多流行的数据库设计软件，如 PowerBuilder、Delphi、VB 等来缩短开发时间并使系统设计更优化。

PowerDesigner 系列产品提供了一个完整的建模解决方案，业务或系统分析人员、设计人员、数据库管理员 DBA 和开发人员可以对其裁剪以满足他们的特定的需要；而其模块化的结构为购买和扩展提供了极大的灵活性，从而使开发单位可以根据其项目的规模和范围来使用他们所需要的工具。PowerDesigner 灵活的分析和设计特性允许使用一种结构化的方法有效地创建数据库或数据仓库，而不要求严格遵循特定的方法学。PowerDesigner 提供了直观的符号表示使数据库的创建更加容易，并使项目组内的交流和通信标准化，同时能更加简单地向非技术人员展示数据库和应用的设计。

PowerDesigner 不仅加速了开发的过程，也向最终用户提供了管理和访问项目信息的一种有效的结构。它允许设计人员不仅创建和管理数据的结构，而且开发和利用数据的结构，针对领先的开发工具环境快速地生成应用对象和数据敏感的组件。开发人员可以使用同样的物理数据模型查看数据库的结构和整理文档，以及生成应用对象和在开发过程中使用的组件。应用对象生成有助

于在整个开发生命周期提供更多的控制和更高的生产率。

（2）PowerDesigner 的模块

PowerDesigner 包含 6 个紧密集成的模块，允许个人和开发组的成员以合算的方式最好地满足他们的需要。这 6 个模块如下。

- PowerDesigner ProcessAnalyst

用于数据分析或数据发现。ProcessAnalyst 模型易于建立和维护，并可用在应用开发周期中确保所有参与人员之间顺畅地通信。这个工具使用户能够描述复杂的处理模型以反映他们的数据库模型。通过表示这些在系统中的处理和描述它们交换的数据，使用 ProcessAnalyst 可以以一种更加自然的方式描述数据项。

- PowerDesigner DataArchitect

用于两层即概念层和物理层的数据库设计和数据库构造。DataArchitect 提供概念数据模型设计，自动的物理数据模型生成，非规范化的物理设计，针对多种数据库管理系统（DBMS）的数据库生成，开发工具的支持和高质量的文档特性。使用其逆向工程能力，设计人员可以得到一个数据库结构的“蓝图”，用于文档和维护数据库或移植到一个不同的 DBMS。

- PowerDesigner AppModeler

用于物理数据库的设计和应用对象及数据敏感组件的生成。通过提供完整的物理建模能力和利用那些模型进行开发的能力，AppModeler 允许开发人员针对领先的开发环境，包括 PowerBuilder、Visual Basic、Delphi 2.0 和 Power++，快速地生成对象和组件。此外，AppModeler 还可以生成用于创建数据驱动的 Web 站点的组件，使开发人员和设计人员同样可以从一个 DBMS 发布“动态”的数据。另外，AppModeler 提供了针对超过 30 个 DBMS 和桌面数据库的物理数据库生成、和维护和文档。

- PowerDesigner MetaWorks

通过模型的共享支持高级的团队工作的能力。这个模块提供了所有模型对象的一个全局层次结构的浏览视图，以确保贯穿整个开发周期的一致性和稳定性。MetaWorks 提供了用户和组的说明定义以及访问权限的管理，包括模型锁定安全机制。它还包含 MetaBrowser，一个灵活的字典浏览器，用以浏览、创建和更新跨项目的所有模型信息和 Powersoft ObjectCycle，一个版本控制系统。

- PowerDesigner WarehouseArchitect

用于数据仓库和数据集市的建模和实现。WarehouseArchitect 提供了对传统的 DBMS 和数据仓库特定的 DBMS 平台的支持，同时支持维建模特性和高性能索引模式。WarehouseArchitect 允许用户从众多的运行数据库引入（逆向工程）源信息。WarehouseArchitect 维护源和目标信息之间

的链接追踪，用于第三方数据抽取和查询及分析工具。WarehouseArchitect 提供了针对所有主要传统 DBMS，诸如 Sybase、Oracle、Informix 和 DB2，以及数据仓库特定的 DBMS（如 Red Brick Warehouse 和 ASIQ）的完全数据仓库处理支持。

- PowerDesigner Viewer

用于以只读的、图形化的方式访问建模和元数据信息。Viewer 提供了对 PowerDesigner 所有模型信息的只读访问，包括处理、概念、物理和仓库模型。此外，它还提供了一个图形化的查看模型信息的视图，Viewer 提供了完全的跨所有模型的报表和文档功能。

（3）PowerDesigner 的通用特性

- 需求管理

PowerDesigner 可以把需求定义转化成任意数量的分析及设计模型，并记录需求和所有分析及设计模型的改动历史，保持对它们的跟踪。Microsoft Word 导入/导出功能使业务用户能轻易处理流程工作。

- 文档生成

PowerDesigner 提供了 Wizard 向导协助建立多模型的 RTF 和 HTML 格式的文档报表。项目团队中非建模成员同样可以了解模型信息，增强整个团队的沟通。

- 影响度分析

PowerDesigner 模型之间采用了独特的链接，与同步技术进行全面集成，支持企业级或项目级的全面影响度分析。从业务过程模型、UML 面向对象模型到数据模型都支持该技术，大大提高了整个组织的应变能力。

- 数据映射

PowerDesigner 提供了拖放方式的可视化映射工具，方便、快速及准确地记录数据依赖关系。在任何数据和数据模型 数据与 UML 面向对象模型以及数据与 XML 模型之间建立支持影响度分析的完整的映射定义、生成持久化代码以及数据仓库 ETL 文件。

- 开放性支持

PowerDesigner 支持所有主流开发平台：支持超过 60 种（版本）关系数据库管理系统，包括最新的 Oracle、IBM、Microsoft、Sybase、NCR Teradata、MySQL 等，支持各种主流应用程序开发平台，如 Java、J2EE、Microsoft.NET（C#和 VB. NET）、Web Services 和 PowerBuilder，支持所有主流应用服务器和流程执行语言，如 ebXML 和 BPEL4WS 等。

- 可自定义

PowerDesigner 支持从用户界面到建模行为以及代码生成的客户化定制。支持用于模型驱动

开发的自定义转换，包括：对 UML 配置文件的高级支持、可自定义菜单和工具栏、通过脚本语言实现自动模型转化、通过 COM API 和 DDL 实现访问功能以及通过模板和脚本代码生成器生成代码。

- 企业知识库

PowerDesigner 的企业知识库是存储在关系数据库中的完全集成的设计时知识库，具有高度的可扩展性，便于远程用户使用。该知识库提供以下功能：基于角色的模型和子模型访问控制，版本控制和配置管理、模型与版本的变更报告以及全面的知识库搜索功能。PowerDesigner 的知识库还可以存储和管理任何文档，包括 Project 文件、图像和其他类型的文档。

2. ERWin

（1）ERWin 介绍

ERWin 的全称是 AllFuusin ERwin Data Modeler，是 CA 公司 AllFuusin 品牌下的数据建模工具。支持各主流数据库系统。其设计图支持 MS Office 的直接拷贝。

（2）ERWin 的主要特点

- 结构复杂数据的可视化

ERWin 提供数据库结构，管理界面简单。

- 设计层架构

ERWin 提供了独特的灵活性，从逻辑、物理，甚至更高级别类型出发以创建多个模型层。用户可以创建完全分开的逻辑和物理模型，或者创建逻辑和物理模型有关联，让用户来选择最适合的实现风格。

- 标准的定义

可重复使用的标准，提高组织的开发能力并有效地管理时间。通过可重复使用的模型模板、域编辑器、命名标准编辑器和数据类型标准编辑器、ERWin 支持标准的定义和维护。

- 大型模型管理

ERWin 通过主题领域的大型企业级模型来帮助管理图表。这些图形的意见和模型可视化，促进不同利益相关者和组织在信息交流方面的合作。此外，先进的功能如自动版式、按需 UI 组件，可使用户轻松地建立大型可视化模型。

- 数据库设计的一代

ERWin 允许用户直接从视觉模型创建数据库设计从而提高效率，减少错误。业界领先的数据库支持，包括优化的参照完整性触发器模板和丰富的跨数据库宏语言，使建模人员定制触发、脚本和存储过程。自定义的模板有助于建立一个模型的完整的物理设计与下一代的定义。

- 数据仓库和数据集市设计

ERWin 支持数据仓库的特定模式（如星型模型和雪花三维建模）技术，从而优化了分析需要的数据仓库。它还捕捉有关的仓库信息，包括数据源、转换逻辑和数据管理规则。

5.4 本章小结

本章主要对数据的查询、分析和建模技术进行介绍。随着互联网时代数据的爆炸式增长，海量数据相关的处理已经成为人们研究的重点之一。海量数据的查询主要是针对其数据量大，数据格式多样化，以及数据价值密度低的特点，对数据中有价值的数据进行查询、提取，寻找用户感兴趣的数据。数据分析针对海量数据，对海量数据中有价值的数据进行提取分析，总结规律，在数据的基础上形成数据价值，为用户所用。数据建模技术是对已有的海量数据的存储及索引进行建模，优化存储与查询结构，使得在数据的存储以及使用上效率提高。

在本章中，通过对数据查询、分析和建模中所使用的相关技术的介绍，直观地展现了当今数据查询、分析和建模技术的发展。并通过介绍数据查询、分析以及建模技术的相关技术与工具展示海量数据处理的发展与现状。

第二篇

大数据深入篇



第 6 章

采用OSGi框架构建可伸缩的异构数据采集平台

大数据时代，信息量庞大，数据承载方式也千变万化，异构数据给互联网带来更多色彩的同时，也给抽取、存储、分析造成了一定的阻碍，例如时下流行的社交网络、微博、博客、RSS 新闻聚合以及各类数据服务，如何把这些异构数据抽象整合，是本章着眼的核心问题。

本章将以一个舆情监控系统的异构数据采集平台为实例，介绍 OSGi 的基本构建方法，并围绕该异构数据采集平台的设计与实现，对 OSGi 在构建动态部署的可伸缩的异构数据采集平台中的设计和使用进行介绍。

6.1 应用背景

本章项目为一个舆情监控系统的异构数据采集平台。

舆情监控系统整合互联网信息采集技术及信息智能处理技术，通过对互联网海量信息自动抓取、自动分类聚类、主题检测、专题聚焦，实现用户的网络舆情监测和新闻专题追踪等信息需求，形成简报、报告、图表等分析结果，为客户全面掌握群众思想动态，做出正确舆论引导，提供分析依据。

舆情监控的首要过程即为信息采集，而采集前端需要接入各种存储方式、组织结构、信息形态都不尽相同的数据平台，利用爬虫等逻辑自动抓取各方信息，而后就需要对这些异构的数据进行解析处理，以供最后的自动分类聚类、主题检测、专题聚焦使用，实现对网络舆情监测和新闻专题追踪等信息需求。

图 6.1 给出了舆情监控系统采集平台的主要流程。

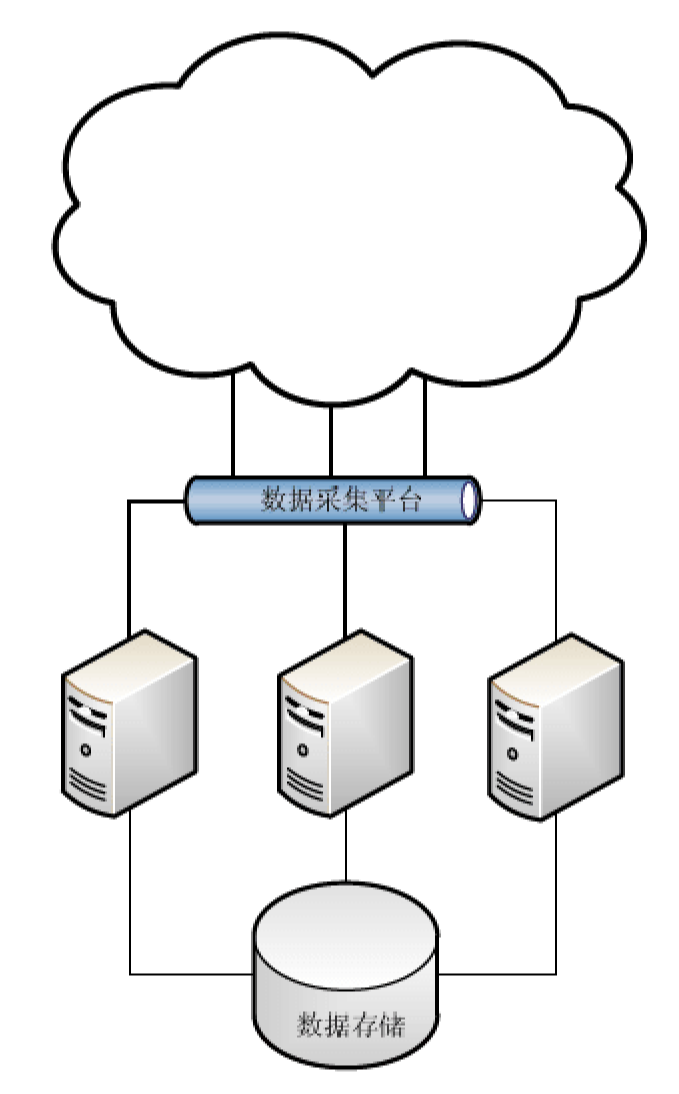


图 6.1　舆情监控数据采集平台流程

该项目需要实现一个适应多平台异构数据不断变化的需求的采集平台，因而必须满足高扩展、可伸缩的特性，所以采用插件的机制对不同数据提供不同的抓取方式，并且通过标准数据接口的形式，对数据采集后的解析处理结果进行约束规范，得到统一同构的数据类型，以便于舆情监控系统后续的分析处理。

图 6.2 给出了本章项目异构数据采集平台的使用方式。

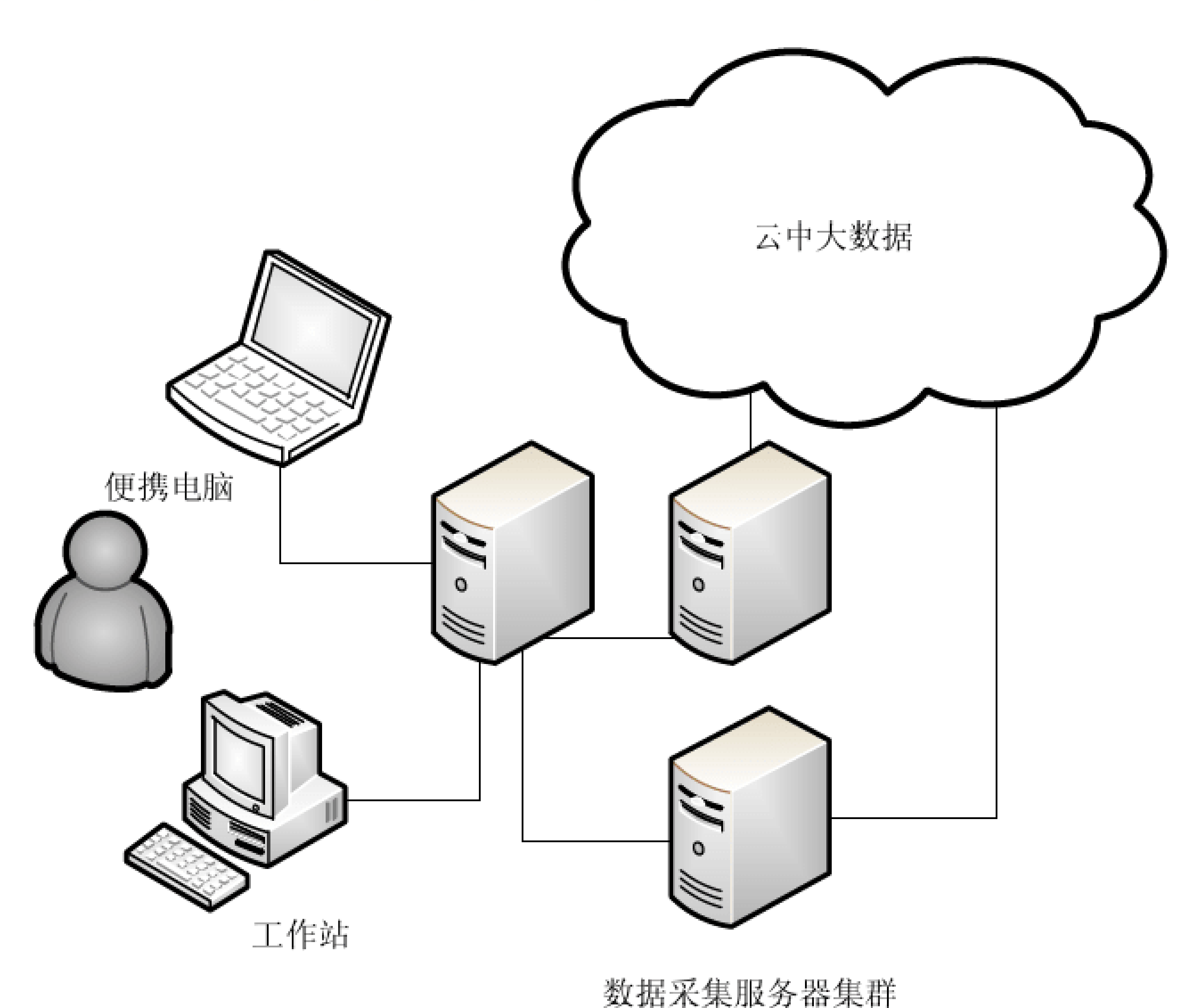


图 6.2　异构数据采集平台的使用方式

舆情监控系统用户通过浏览器，使用各类设备接入系统，对系统中的数据采集集群服务器进行操作，并按照异构数据的需求，定制舆情信息的采集插件，通过满足约定的插件，可以采集到统一的舆情信息，存储于数据库中，供后续的分析处理使用。

6.2 需求分析与总体设计

根据上一节内容的项目应用背景，我们针对现有舆情监控系统对数据采集的需求做了调查，在此基础上发现，不同的采集系统或算法有其不同的实现方式，对不同的 Web 页面也有不同的抽取效果与效率，所以各有其优缺点以及应用的范围。

6.2.1　功能需求

为了满足对互联网大数据背景下异构数据的处理需求，并且解决不断变化的数据格式带来的系统部署不便等问题，本平台的功能在覆盖了完整的数据抽取、解析与存储流程的同时，提供了对多平台数据采集进行管理和动态部署的一套工具。

异构数据抽取和存储平台，其核心功能即为数据抽取与存储。本平台为了实现可伸缩、可扩展的插件机制需要引入插件框架来处理各模块间的调度控制功能。

采集平台的主要功能分为以下部分。

- 数据抽取：对不同平台数据进行抽取，提供相关安全授权、验证接入的功能。
- 数据解析：对抽取得到异构数据解析后输出满足存储规定格式的同构数据。
- 数据存储：对解析后数据提供存储、持久化、合并数据库等功能。
- 插件管理：处理以上功能模块间的监控调度控制。

如图 6.3 所示为本项目平台的主要功能组织结构。

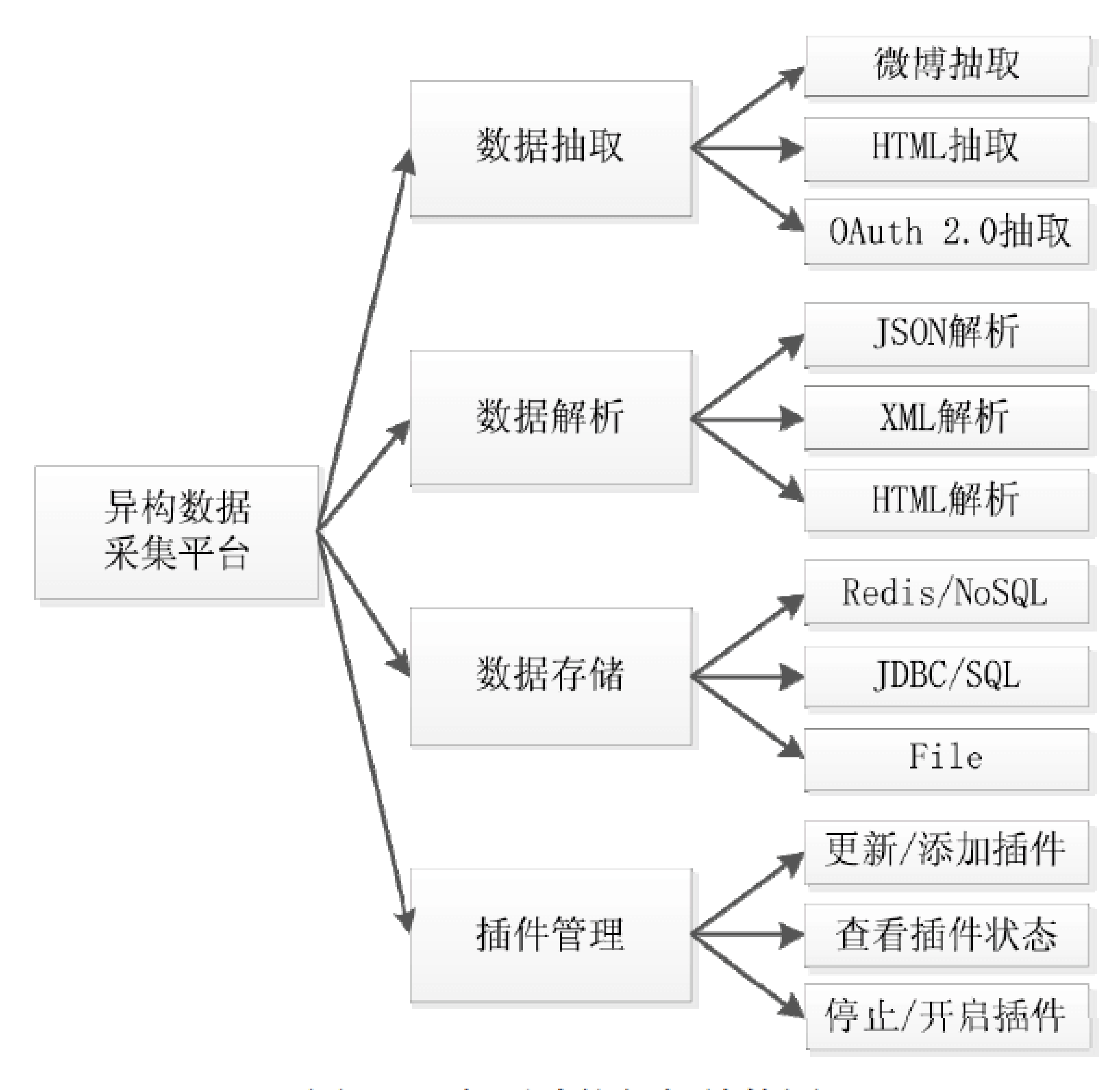


图 6.3　主要功能组织结构图

- 数据抽取部分，微博、OAuth 2.0、HTML 等功能插件为本平台抽取数据所需的基础功能，

其作用是为用户自定义抽取逻辑提供验证接口以简化获得原始文本的操作。

- 数据解析部分，可调用各类文本格式的解析库，以此为抽取过程中的解析操作提供支持。
- 数据存储部分，可以选择 NoSQL、SQL、文件等多种方式存储，以应变不同的分析环境，该功能模块也提供了相应的数据库管理系统的接口，以简化数据库的接入。
- 插件管理部分，可以监视插件运行状态，并且管理员能够通过发送指令控制插件的开启和关闭，并能添加或更新新插件。

图 6.3 即为本采集平台的主要功能，其中数据抽取、解析、存储部分的调度功能由插件管理功能模块来处理。

6.2.2 非功能需求

为了应对大数据背景下的异构数据采集需求，舆情监控系统的数据采集平台开发普遍存在两方面的困难。

一方面，多源异构数据，即来自不同平台的非统一化结构数据，这类数据在网络发达的今天普遍存在，例如 HTML、XML、JSON、数据库接入服务等，它们不管是获取方式还是存储格式、不同的厂商在不同的场合、选取的数据结构，都大相径庭，因此带来了数据抽取和存储上的难题，如何设计抽取存储平台，把异构数据整合统一格式化，以供后续的分析处理，是大数据技术不得不考虑的问题。

另一方面，根据一定的先验知识产生包装器的方法造成了系统的适应性较差，即通用性不够，当网页结构发生变化时，需要重新进行人工干预，对抽取工作造成了一定的困难，不断关闭系统再重新上线部署新的算法逻辑的同时也对系统的可用性造成了影响。

这两个方面的问题，都把矛头指向了日新月异的技术和需求变化带来的数据异构状况，而为了适应这种快速变化，我们需要一个可伸缩、扩展能力强的平台来解决从开发、更新到部署、使用流程中遇到的一系列困难。

因此本章数据采集平台的设计与实现，考虑到上述问题，主要强调对多种异构数据的灵活接入。

6.2.3 总体设计

异构数据采集平台以数据为核心，因此在设计数据流时需要考虑的问题比较多。根据以上需求分析获得的用例规约，把整个系统分作三个核心模块和一个管理模块，其中三个核心模块由系统定义接口，并由插件开发人员实现具体逻辑，管理模块采用 RESTful 数据获取模式，可以方便地组建自定义的管理后台。

如图 6.4 所示是系统的上下文数据流图。

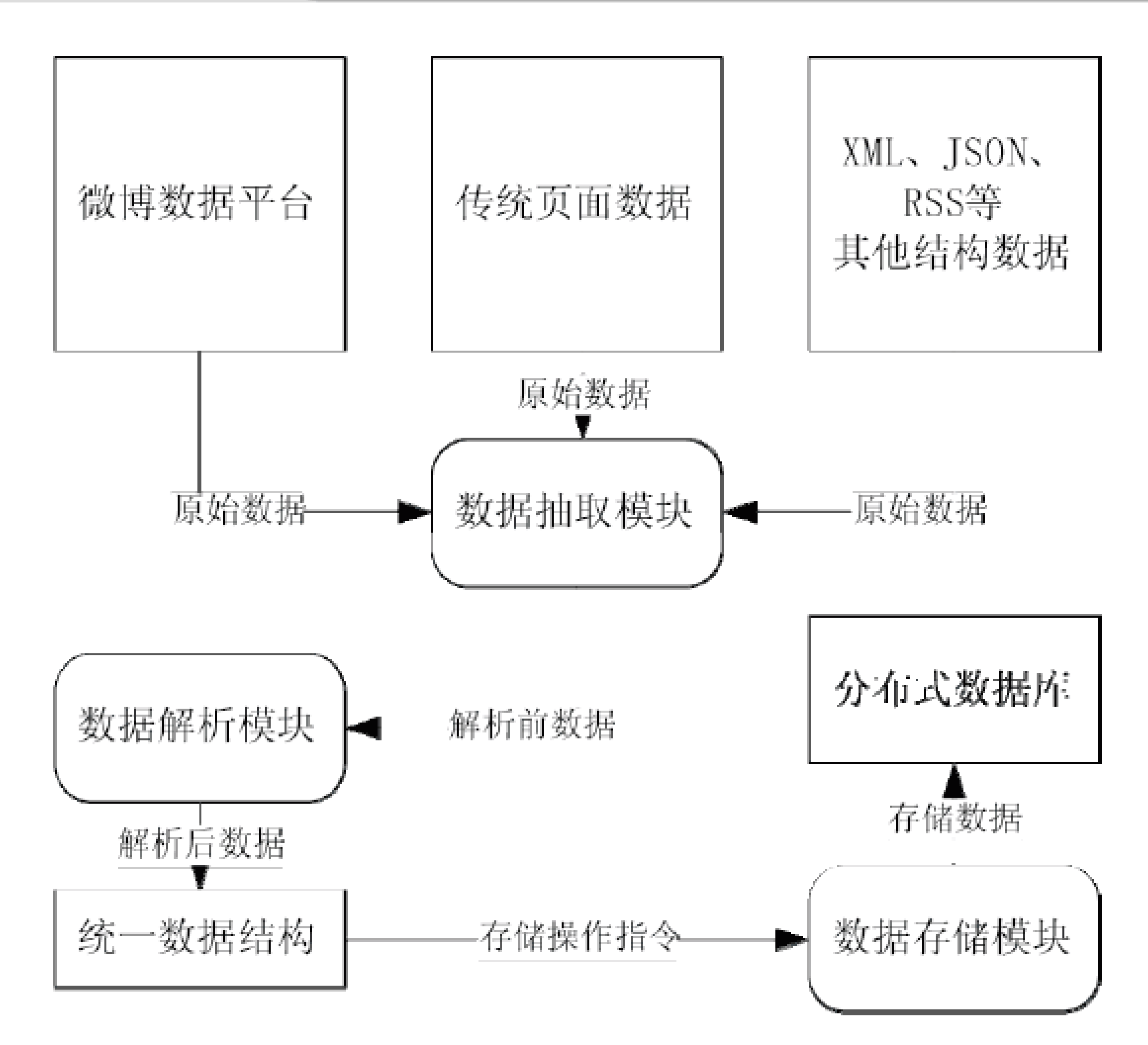


图 6.4　系统上下文数据流图

由于需要抽取异构数据，因而各功能模块间数据流采用管道过滤器模式。

各模块生命周期如图 6.5 所示。

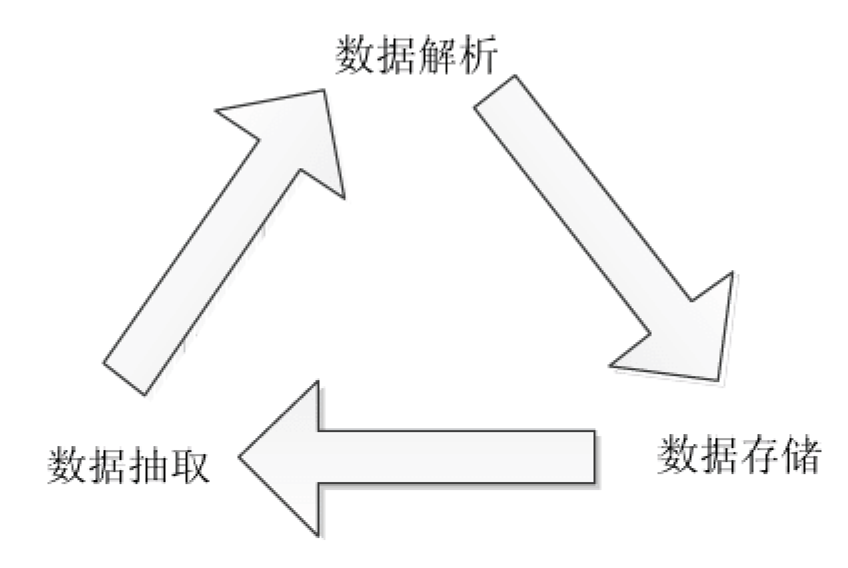


图 6.5　插件模块生命周期图

对原始数据源进行访问获取后，以当前页面或用户等数据键为节点，通过解析器解析其中的外部链接，以此为边向外爬取新数据，在每个节点爬取过程中，将采集得到的元数据解析为对象，按网状结构存储到数据库。这一系列操作的枢纽就是异构数据到同构数据的转化。

采集平台系统总架构图如图 6.6 所示。本框架采用基于 OSGi 规范的插件式框架，之所以使用 OSGi 作为数据抽取和存储的插件平台，主要考虑到其动态绑定部署的特性，能够实现可伸缩的插件机制，以此同时抓取更多平台（新浪、腾讯、Twitter……），可以纳入更多的异构数据和接入支持（HTML、XML、JSON、HTTP、HTTPS、SQL……），并且每个抽取插件相对独立，支持分布式和动态部署，能够针对不同地区的网络环境选择不同服务器来完成信息采集，利用 Redis 等内存数据库减少磁盘 IO 等。

本平台还需要实现一个控制台工具，对本平台的各插件的运行情况作监视，同时使用该工具可以实现动态部署增加新功能、开启关闭插件等操作，该工具采用 B/S 架构，Node.js 与 Felix（OSGi 框架的一个流行实现）的 console 交互，封装对插件管理“增加、更新、开关”的功能，并可供远程访问。

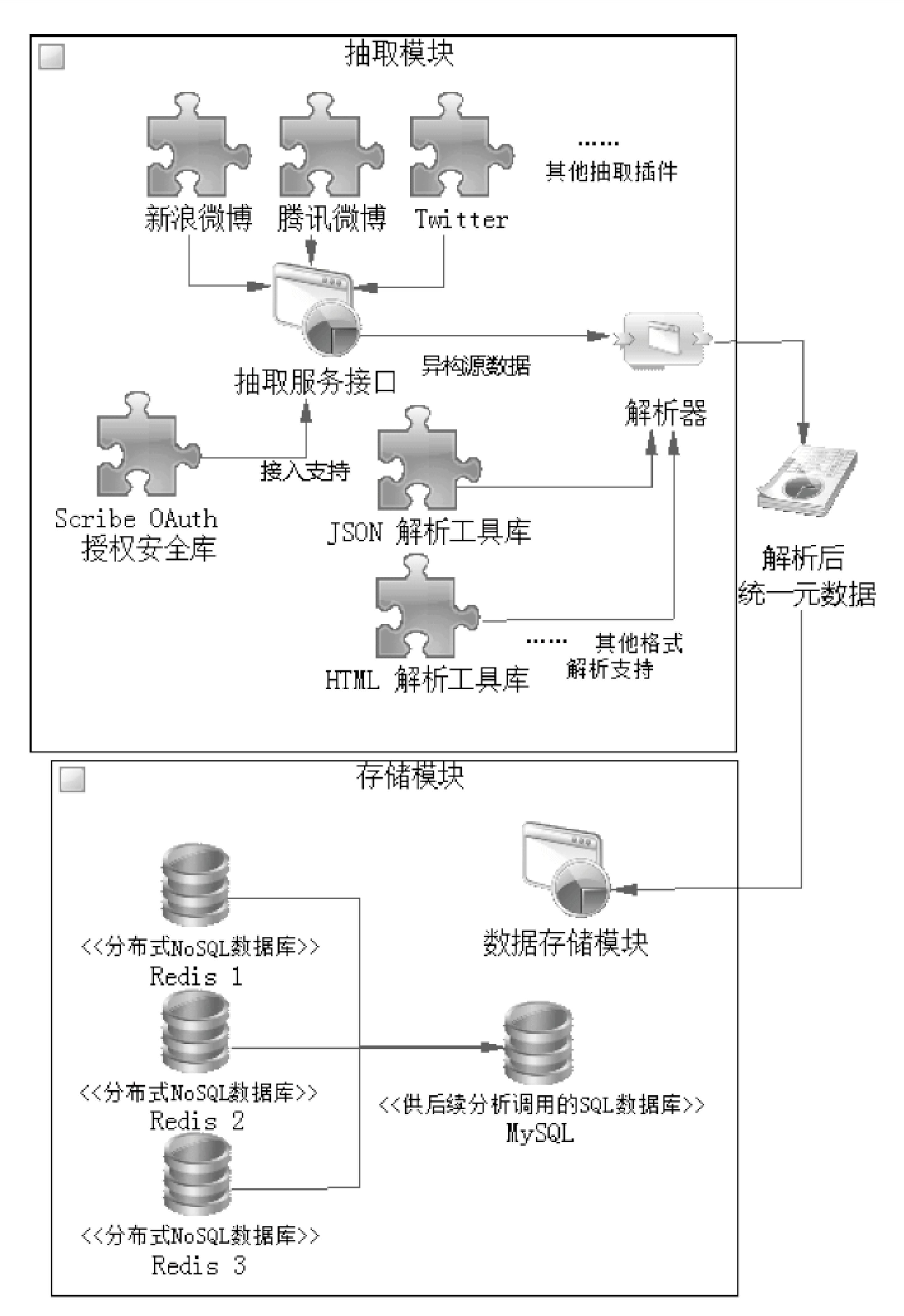


图 6.6　采集平台系统总体架构图

6.3 相关技术介绍

6.3.1 OSGi 框架介绍

OSGi（Open Service Gateway Initiative）开放服务平台技术是面向 Java 的动态模型系统。OSGi 服务平台向 Java 提供服务，这些服务使 Java 成为软件集成和软件开发的首选环境。Java 提供在多个平台支持产品的可移植性。OSGi 技术允许应用程序使用精炼、可重用和可协作的组件构建的标准化原语。这些组件能够组装进一个应用和部署中。

1. OSGi 框架要点

OSGi 框架形成了 OSGi 规范的核心。它提供了一个通用、安全、受管理的 Java 框架，为可扩展和可下载的应用或服务集合（以下被称作 Bundle）的部署提供支持。

OSGi 兼容的设备，可以下载并安装 OSGi Bundle 包，并当不再需要时，删除它们。该框架在 OSGi 的环境中，以动态、可伸缩的方式安装和更新管理 Bundle。为了实现这一目标，它需要管理详细的 Bundle 和服务之间的依赖关系。

它提供给 Bundle 开发者必要的资源，以充分利用 Java 的平台独立性和动态代码装载能力，来轻松地开发为小内存设备大规模部署的服务。

如图 6.7 所示为 OSGi 框架的功能被划分的层。

- 执行环境（Execution Environment）
- 安全层（Security）
- 模块层（Module）
- 生命周期层（Life cycle）
- 实际服务层（Service）

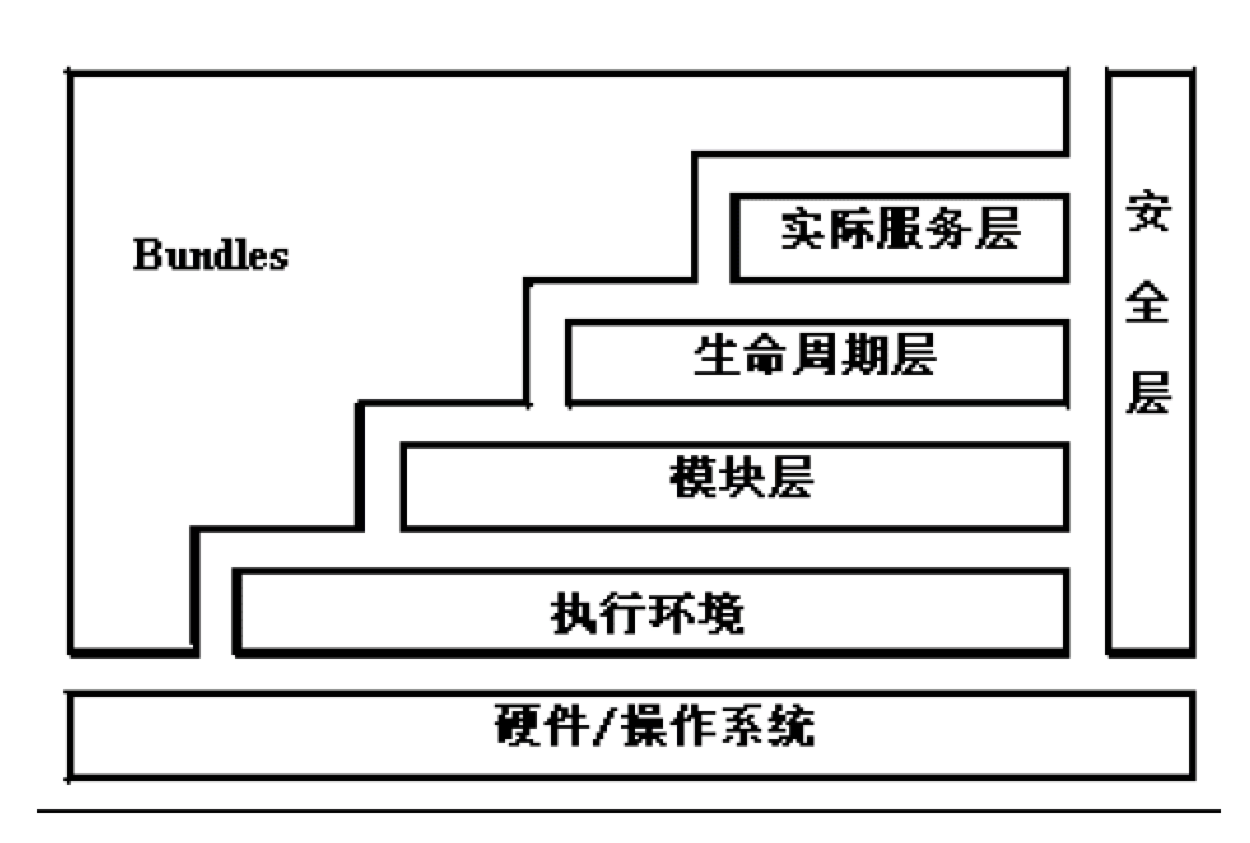


图 6.7　OSGi 主要架构图

通过以上的功能分层，在 Java 的 Runtime 运行时环境中，满足 OSGi 规范约定的 Bundles 可以在框架的管理下，动态地增加、更新和移除服务提供类，这一点正是构建可伸缩插件平台的特性。

2. OSGi 技术细节

（1）OSGi Headers

OSGi Headers 是存放在 jar 包中的 Manifest 头文件，是 OSGi 框架对 Bundle 的信息入口。不同的组织为 OSGi Bundles 提供了许多 Header 元信息。

主要包括：

- Bundle-SymbolicName : Bundle 包名。
- Bundle-Version : Bundle 的版本号。

- Export-package：导出包，指明对外可用的包。
- Import-package：引入包，指明绑定依赖的包。

（2）OSGi 生命周期

OSGi 生命周期包括三个状态，即 Installed（已安装）、Resolved（待机）和 Uninstalled（已卸载）。

其中 Resolved 包含 Starting、Active 和 Stoping 三个受事件驱动的执行状态。

OSGi 框架在成功加载 Bundles 后，如果 Bundle 内包含 Activator 类型，则可以通过外部命令调用该类中声明的 start 和 stop 函数，来实现服务的开启和关闭。

而当需要安装或者移除 Bundle 时，也提供了 install/update、uninstall 选项。

（3）Bundle Activator

每个 Bundle 能够可选地声明一个实现了 org.osgi.framework.BundleActivator 的 Activator（催化剂）类，这个类必须在 Manifest 文件中被 BundleActivator 引用，通过实现这个类，允许 Bundle 开发者在生命周期中实现具体的 start 和 stop 动作，一般来说，该类操作包括获取和释放资源、注册和注销服务等。

3. OSGi 的动态绑定特性

（1）特性介绍

OSGi 框架提供了一个非常强大和动态的程序环境，Bundles 的安装、启动、停止、更新、卸载都不需关停框架。Bundles 的依赖由框架监控，但 Bundles 必须保证这些依赖关系正确。框架的动态特性关乎两个重要方面，那就是“服务注册”和“已安装的 Bundles 集合”。

Bundles 开发人员必须注意，避免使用已注销的陈旧的服务对象。框架服务注册的动态性质，使我们有必要跟踪服务对象，判断它们是注册还是未注册的，以此防止依赖问题。我们很容易忽视其中的竞争或边界情况，这将导致某些不确定的错误。跟踪已安装的 Bundle 集合和它们的状态时也存在类似的问题。

为此，OSGi 规范定义了两个工具类，ServiceTracker 和 BundleTracker，这使得跟踪服务和捆绑更加容易。

一个 ServiceTracker 的类可以通过实现 ServiceTrackerCustomizer 接口或继承 ServiceTracker 类的子类来定制功能。同样的，一个 BundleTracker 类也可以通过实现 BundleTrackerCustomizer 接口或子类来定制。这些实用工具类，显著地降低了跟踪“服务注册”和“已安装的 Bundle 集合”的复杂度。

（2）跟踪服务和 Bundles

OSGi 框架是一个动态的多线程环境。在这样的环境中，回调可以同时发生在不同的线程中。这种动态性导致了更多的复杂性。实现这种环境出奇困难的方面之一就是如何加强“跟踪服务和 Bundles”的可靠性。这些复杂问题引发自 BundleListener 和 ServiceListener 接口在 Listener 监听器注册时只提供状态变更的访问权，而不是提供完整的现有服务状态。这给程序员留下了一系列问

题——如何将以事件为标志的状态变更，与现有服务状态合并起来，并且规避那些“服务冗余”或者“因缺失了某个 remove 事件，导致实际不存在的服务仍在被框架跟踪”之类的问题。

通过合适的锁结构，可以缓解这些易发生的竞争和边界问题，但是也容易引发死锁，因此，OSGi 框架提供了 Service Tracker 和 Bundle Tracker，显著减少了解决这一系列问题的难度。

Bundle 其实就是一个 jar 文件，这个 jar 文件和普通的 jar 文件唯一不同的地方就是 META-INF 目录下的 MANIFEST.MF 文件的内容，关于 Bundle 的所有信息都在 MANIFEST.MF 中进行描述，说得时髦点，可以称它为 Bundle 的元数据，这些信息中包含有类似 Bundle 的名称、描述、开发商、classpath 、需要导入的包以及输出的包等，在后续的开发 Bundle 中将会详细介绍 Bundle 的元数据以及如何去开发 Bundle。

Bundle 通过实现 BundleActivator 接口来控制其生命周期。Bundle 启动、停止时所需要进行的工作可以在 Activator 中编写，也可在 Activator 中发布或者监听框架的事件状态信息，根据框架的运行状态做出相应的调整。但同时要注意，如果应用是被类似于 Ctrl+C 等方式强行终止的话，那么 Activator 中的 stop 方法是不会被调用的。

Bundle 是一个独立的概念，在 OSGi 框架中对于每个 Bundle 采用的是独立的 classloader 机制，这也就意味着不能采用传统的如引用其他 Bundle 的工程来实现 Bundle 间的协作。那么在 OSGi 框架中 Bundle 之间是怎么协作的呢？在 OSGi 框架中对于每个 Bundle 可以定义输出的包以及引用的包，这样在 Bundle 间就可以共享包中的类了，尽管这样也可以直接实现简单的 Bundle 的协作，但在 OSGi 框架中更加推荐的是采用 Service 的方式，Service-Oriented 的概念（例如 SOA）大家都接触多了，OSGi 框架也同样如此，每个 Bundle 可以通过 BundleContext 注册对外提供的服务，同时也可以通过 BundleContext 来获得需要引用的服务，采用 Service-Oriented 的方式可以使得对外提供的服务更加封闭，不需要为了使用别的 Bundle 提供的 Service 而做环境依赖等的设置，同时，Bundle 还可以采用 Require-Bundle 的方式来直接引用其他的 Bundle（相当于引用其他 Bundle 的工程或 jar），在后续的开发、发布和使用 Service 中将会详细介绍如何去开发、部署和使用 Service。

（3）Bundle 上下文

当 Bundle 被启动时，框架调用其激活器的 start()方法，当它被停止时，则调用 stop()方法。这两个方法都会接收 BundleContext 接口的一个实例。BundleContext 接口的方法可以大致分为两类。

第一类与部署和生命周期管理相关。第二类与 Bundle 间服务式的交互相关。我们对第一类提供了许多额外的能力，包括安装和管理其他 Bundle 的生命周期、获取框架的有关信息、查找基本配置属性。

Bundle 上下文对象是与之关联的 Bundle 唯一执行时的上下文环境，它的这种角色是它的一个重要方面。因为它代表了执行上下文，所以只有当关联的 Bundle 处于激活状态时，它才是有效的。更明确地说，它是指，从激活器的 start()方法被调用，到 stop()方法结束的这一整段时间内。当关联的 Bundle 没有处于激活状态时，试图使用它，大多数 Bundle 上下文的方法都会抛出异常。它是唯一的执行时上下文环境，这是由于每个已激活的 Bundle 都会接收到属于自己的上下文对

象。框架之所以使用该上下文是出于安全性，以及为每个独立的 Bundle 分配资源的考虑。基于 BundleContext 对象的这一特性，它们应被视为敏感的或私有的对象，并且不能在 Bundle 之间自由传递。

（4）Service 依赖管理系统

OSGi 服务的松散结构，使得开发中对 Service 的依赖变得复杂。这也使得 Service 的关系管理成为 OSGi 中一个非常重要的部分，本章中使用了 OSGi Service 依赖关系管理的一种方式——Service Listener。

这是 OSGi 中原生的 Service 依赖管理机制，也是最简单直接的方式。

标准的注册/查找步骤如下：

步骤01 被依赖的 Bundle 通过 BundleContext.registerService()方法注册服务到系统中。

步骤02 使用依赖的 Bundle 在 start 时通过 BundleContext 的 getServiceReferences()/getService()来查找依赖的 service。

步骤03 使用依赖的 Bundle 通过 BundleContext.addServiceListener()来一一注册 ServiceListener。

步骤04 在被依赖的 Bundle/service 状态发生变化时，使用依赖的 Bundle 通过 ServiceListener 调用 serviceChanged()得到通知并作出调整。

在这种方法中，使用依赖的 Service 必须进行大量的编码工作来完成对依赖的 Service 进行关系管理，需要处理琐碎细节（如各个 Service 运行时的状态变化）。为了减少工作量，OSGi 设计了 ServiceTracker 来简化对依赖 Service 的编码工作，即 ServiceTracker 将负责处理上述步骤中的步骤02、步骤03、步骤04。

经过 ServiceTracker 优化后的 Service Listener 机制，还是存在一些缺点：

- 编码量还是不小，尤其对于依赖较多的场景。
- Activator 还是太复杂了，尽管已经努力地试图简化。但对于一些业务逻辑简单的 Service，如果依赖的 Service 比较多，那么 Activator 的逻辑和代码实现远比 Service 本身的逻辑和实现要复杂，这违背了我们使用框架简化开发的初衷。
- Activator 对测试不利，这是 Activator 的复杂性造成的。由于 Activator 中存在大量的依赖处理逻辑，理所当然地会增加测试的复杂性。

总的来说，Service Listener 机制下，管理 Service 依赖对于开发者来说难度较大：管理必不可少，但又容易出现错误，出错时不容易测试。而且，这些工作都不是 Service 业务逻辑的组成部分，不能带来直接收益。更重要的是，从分工的角度上讲，开发人员应该将更多的精力投入到应用层与逻辑层，而不是 OSGi 的底层实现机制。

6.3.2 多源异构数据的获取

本章限于篇幅，将介绍互联网领域的 SNS、微博、HTML 等异构数据的抽取存储设计与实现。

1. 常见异构数据抽取方式

（1）RESTful API 与 OAuth 2.0 授权机制

REST（REpresentational State Transfer）是一组架构约束条件和设计原则，REST 描述了一个架构样式的网络系统，比如 Web 应用程序。在服务器端，应用程序状态和功能可以分为各种资源。资源是一个有趣的概念实体，它向客户端公开。资源的例子有：应用程序对象、数据库记录、算法等。每个资源都使用 URI（Universal Resource Identifier）得到一个唯一的地址。所有资源都共享统一的界面，以便在客户端和服务器之间传输状态。它使用的是标准的 HTTP 方法，比如 GET、PUT、POST 和 DELETE。Hypermedia 是应用程序状态的引擎，资源表示通过超链接互联。

许多 SNS 与微博，都提供了一套完善的满足 REST 约束的开放数据平台，这类平台通常会提供结构化的纯净数据，凡是满足 OAuth 用户授权的应用，都可以通过 HTTP 来获取相关数据。我们可以通过该类平台，来实现微博的数据抽取。

OAuth 允许用户提供一个令牌，而不是用户名和密码来访问受保护的数据。每一个令牌授权一个特定的网站（例如，视频编辑网站）在特定的时段（例如，接下来的 2 小时内）内访问特定的资源（例如仅仅是某一相册中的视频）。这样，OAuth 允许用户授权第三方网站来访问他们受保护的信息，而非共享他们的访问许可或数据的所有内容。

在认证和授权的过程中涉及的三方包括：

- 服务提供方，用户使用服务提供方来存储受保护的资源，如照片、视频、联系人列表。
- 用户，存放在服务提供方的受保护的资源的拥有者。
- 客户端，要访问服务提供方资源的第三方应用，通常是网站，如提供照片打印服务的网站。在认证过程之前，客户端要向服务提供者申请客户端标识。

使用 OAuth 进行认证和授权的过程如下所示：

步骤 01　用户访问客户端的网站，请求用户存放在服务提供方的资源。

步骤 02　客户端向服务提供方请求一个临时令牌。

步骤 03　服务提供方验证客户端的身份后，授予一个临时令牌。

步骤 04　客户端获得临时令牌后，将用户引导至服务提供方的授权页面请求用户授权。在这个过程中将临时令牌和客户端的回调链接网址发送给服务提供方。

步骤 05　用户在服务提供方的网页上输入用户名和密码，然后授权该客户端，访问所请求的资源。

步骤 06　授权成功后，服务提供方引导用户返回客户端的网页。

步骤 07　客户端根据临时令牌从服务提供方那里获取访问令牌。

步骤 08　服务提供方根据临时令牌和用户的授权情况授予客户端访问令牌。

步骤 09　客户端使用获取的访问令牌访问存放在服务提供方上的受保护的资源。

（2）静态页面数据

RESTful 设计原则的 Web 应用，提供了方便快捷的数据抽取方法。但是实际情况下，更多的数据源，并没有满足 REST 约束，也没有提供方便的获取结构化数据的途径。在这种情况下就需

要对原始数据页面进行解析，并通过内部锚链接的跳转来实现爬虫，从而更高效地抽取这些原始页面中的关键数据内容。这类数据中最常见的就是 HTML 与 RSS，该类页面可以很容易地从互联网中获取。但是为了将其关键数据抽取出来，需要做一系列的解析操作，并且需要自行设计数据的组织结构，来满足日后的分析需求。

DOM 是 Document Object Model（文档对象模型）的缩写。根据 W3C DOM 规范，DOM 是一种与浏览器、平台、语言无关的接口，借此可以访问页面其他的标准组件。简单理解，DOM 解决了 Netscape 的 JavaScript 和 Microsoft 的 JScript 之间的冲突，给予 Web 设计师和开发者一个标准的方法，让他们来访问他们站点中的数据、脚本和表现层对象。

DOM 是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中导航寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。DOM 定义了访问和操作文档的标准方法，把 HTML 文档呈现为带有元素、属性和文本的树结构（节点树）。

DOM 分为 HTML DOM 和 XML DOM 两种。它们分别定义了访问和操作 HTML/XML 文档的标准方法，并将对应的文档呈现为带有元素、属性和文本的树结构（节点树）：

- DOM 树定义了 HTML/XML 文档的逻辑结构，给出了一种应用程序访问和处理 XML 文档的方法。
- 在 DOM 树中，有一个根节点，所有其他的节点都是根节点的后代。
- 在应用过程中，基于 DOM 的 HTML/XML 分析器将一个 HTML/XML 文档转换成一棵 DOM 树，应用程序通过对 DOM 树的操作，来实现对 HTML/XML 文档数据的操作。

（3）数据库接入

对于私有数据，尤其是企业内部数据，也会提供相关的数据库接入服务，有可能是满足 SQL 的 ODBC、JDBC 等通用接口或者 MySQL、Oracle 等数据库，也有可能是实现某些特定需求的 NoSQL 数据库。这类数据通常有良好的组织结构，也有便捷的接入和检索方式，因而是最便于后续处理的数据接入方式。

实体关系模型（Entity-Relationship Model），简称 E-R Model 是陈品山（Peter P.S Chen）博士于 1976 年提出的一套数据库的设计工具，他运用真实世界中事物与关系的观念，来解释数据库中抽象的数据架构。实体关系模型利用图形的方式，即实体-关系图（Entity-Relationship Diagram）来表示数据库的概念设计，有助于设计过程中的构思及沟通讨论。

关系模型就是指二维表格模型，因而一个关系型数据库就是由二维表及其之间的联系组成的一个数据组织。当前主流的关系型数据库有 Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL 等。

关系数据库是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。现实世界中的各种实体以及实体之间的各种联系均用关系模型来表示。关系模型是由埃德加·科德于 1970 年首先提出的，并配合“科德十二定律”。如今虽然对此模型有一些批评意见，但它还是数据存储的传统标准。标准数据查询语言 SQL 就是一种基于关系数据库的语言，这种语言执行对关系数据库中数据的检索和操作。关系模型由关系数据结构、关系操作集合、关

系完整性约束三部分组成。

关系型数据库中的表都是存储一些格式化的数据结构，每个元组字段的组成都一样，即使不是每个元组都需要所有的字段，但数据库会为每个元组分配所有的字段。这样的结构可以便于表与表之间的连接等操作，但从另一个角度来说它也是关系型数据库性能瓶颈的一个因素。而非关系型数据库以键值对存储，它的结构不固定，每一个元组可以有不一样的字段，每个元组可以根据需要增加一些自己的键值对，这样就不会局限于固定的结构，可以减少一些时间和空间的开销。

2. 常见异构数据解析

（1）JSON

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。它基于 JavaScript 的一个子集。JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯。这些特性使 JSON 成为理想的数据交换语言，易于人阅读和编写，同时也易于机器解析和生成。

JSON 简单地说就是 JavaScript 中的对象和数组，通过这两种结构可以表示各种复杂的结构。

- 对象：对象在 js 中表示为“{}”扩起来的内容，数据结构为 {key: value,key: value,...}的键值对结构。在面向对象的语言中，key 为对象的属性，value 为对应的属性值，所以很容易理解。取值方法为：对象.key ，这个属性值的类型可以是数字、字符串、数组、对象几种。
- 数组：数组在 js 中是中括号“[]”扩起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言中一样，使用索引获取。字段值的类型可以是数字、字符串、数组、对象几种。

对比如下新浪微博与腾讯微博开放平台的 JSON 数据，可以看出两者的数据封装形式有所不同，因此需要对 JSON 数据对象化后，按照具体数据结构情况处理。

新浪微博数据结构：

```
{//新浪微博时间线
   "statuses": [
      {
         "created_at": "Tue May 31 17:46:55 +0800 2011",
         "id": 11488058246,
         "text": "求关注。",
         "source": "<a href="http://weibo.com" rel="nofollow">新浪微博</a>",
         "favorited": false,
         "truncated": false,
         "in_reply_to_status_id": "",
         "in_reply_to_user_id": "",
         "in_reply_to_screen_name": "",
         "geo": null,
```

```
            "mid": "5612814510546515491",
            "reposts_count": 8,
            "comments_count": 9,
            "annotations": [],
        },
        ...
    ],
    "previous_cursor": 0,                    // 暂未支持
    "next_cursor": 11488013766,             // 暂未支持
    "total_number": 81655
}
```

腾讯微博数据结构：

```
{//腾讯微博时间线
errcode : 0,
msg : ok,
ret : 0,
data :
{
    info : [
    {
        text : 威武、升堂，大人，今天小人有一冤案，你要为我做主呀！，
        origtext : 威武、升堂，大人，今天小人有一冤案，你要为我做主呀！，
        count : 0,
        mcount : 2,
        from :  腾讯手机管家(PC版),
        fromurl : http://m.app.qq.com/android/index.jsp,
        id : 112714089895346,
        image : [http://app.qpic.cn/mblogpic/9c7e34358608bb61a696],
        name : china394337002,
        openid : 8389651A7995DA0D68870BBD1D50D959,
        nick : china,
        self : 0,
        timestamp : 1341054421,
    }],
    user :
    {
        name : nick
    }
```

```
    },
    seqid : 5776011266805341272
   }
```

现有的 JSON 解析工具有很多，几乎在所有常见的语言里都有相应的实现，在 Java 中有 JSON-Lib、org.json、Flexjson 等。

（2）XML 与 HTML

可扩展标记语言 XML，是用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。

超级文本标记语言 HTML 是标准通用标记语言下的一个应用，也是一种规范，一种标准，它通过标记符号来标记要显示的网页中的各个部分。网页文件本身是一种文本文件，通过在文本文件中添加标记符，可以告诉浏览器如何显示其中的内容（如文字如何处理、画面如何安排、图片如何显示等）。

XML 与 HTML 的设计区别是：XML 被设计为传输和存储数据，其焦点是数据的内容。而 HTML 被设计用来显示数据，其焦点是数据的外观。HTML 旨在显示信息，而 XML 旨在传输信息。

XML 和 HTML 的语法区别是：HTML 的标记不是所有的都需要成对出现，XML 则要求所有的标记必须成对出现；HTML 标记不区分大小写，XML 则大小敏感，即区分大小写。

对于该类标记语言数据，通常采用 DOM 解析器来抽取数据。DOM 可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。换句话说，这是表示和处理一个 HTML 或 XML 文档的常用方法。

本章针对该类静态数据，为实现关键数据的抽取，使用 HTML 解析工具库 jsoup。jsoup 是一款 Java 的 HTML 解析器，可直接解析某个 URL 地址、HTML 文本内容。它提供了一套非常省力的 API，可通过 DOM、CSS 以及类似于 jQuery 的操作方法来取出和操作数据。

jsoup 的主要功能如下：

- 从一个 URL，文件或字符串中解析 HTML。
- 使用 DOM 或 CSS 选择器来查找、取出数据。
- 可操作 HTML 元素、属性、文本。

jsoup 是基于 MIT 协议发布的，可放心使用于商业项目。

6.4 系统设计与实现

6.4.1 异构数据采集平台的设计

本章所述的异构数据采集平台，优先针对数据结构做了优化，采用类似中间件的技术，在各个抽取、存储模块间传递信息，能够适应不同的数据接口，灵活应变信息结构变化的需求，由此可以获得一个可伸缩的数据采集平台。

1. 数据模型设计

异构数据的采集工作需要面对不同种类的数据源结构。设计统一的数据模型是建立可伸缩数据采集平台中间层所需要考虑的首要问题。针对现代微博的数据特性，可将其抽象为“用户、feed、关系”三个核心内容：用户信息包含了用户的个人信息、用户的订阅列表、用户之间的关系信息等；feed 包含了具体微博的内容与增量数据、该微博参与用户列表、转发者的信息等；而关系即为用户与用户、用户与 feed、feed 与 feed 之间的联系信息。

在抓取过程中，我们可以把这些数据信息分类为静态和动态的两种。其中用户的关系、订阅列表等为静态信息，而用户订阅的时间线等不断随时间变化的数据则为动态信息。我们把用户与 feed 的关系数据视作静态，保存于数据库中，提供定期更新。而时间线等动态数据则在每一次抓取时都获得更新，按照时间戳比对，由此可以获取新的舆情数据，保证数据的时效性。

```
public class Weibo{
 public String id;
public Content content;
public String timestamp;
public Weibo(String id, Content content) {
     this.id = id;
     this.content = content;
     this.timestamp = System.currentTimeMillis().toString;
}
}
```

如以上代码清单所示，微博的元数据被抽象为包含 id、发布内容、抓取时间戳等内容的类，其中 id 对应具体的数据源，用于防止抽取重复数据造成冗余。发布内容包括发布用户 uid、文本内容、转发信息等，时间戳记录当前抓取时间便于日后检索排序使用。

如图 6.8 所示为数据解析器与微博元数据的关系图。元数据类型中定义了数据的目标结构，通过不同的解析器来对源数据解析并重新组织。最后获得的满足元数据定义的数据类型。

2. 抽取架构设计

如图 6.9 所示为数据抽取模块架构图。

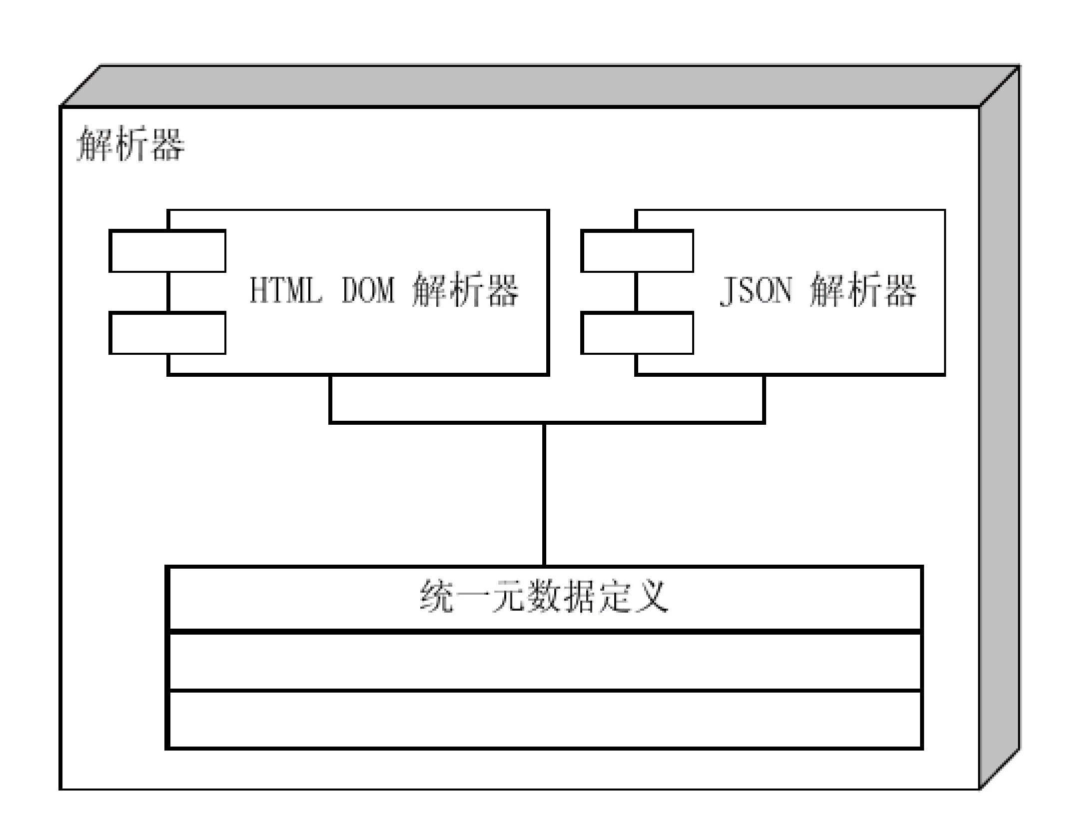


图 6.8　数据解析器与微博元数据的关系图

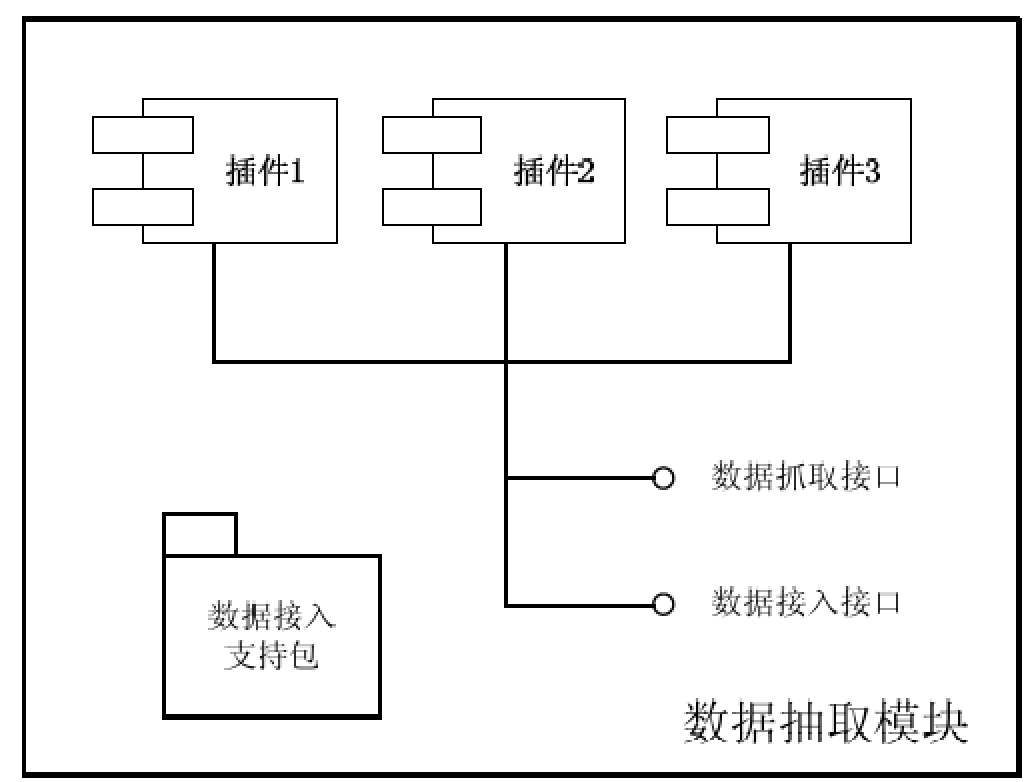


图 6.9　数据抽取模块架构图

系统的抽取架构在设计时，为了便于多平台接入，以及后续的存储处理，首先应当把规范和实现分离。为了抽象数据抽取的模块接口，把抽取架构分为两个部分，其一为数据接入接口，其二为数据抓取逻辑。

（1）数据接入接口

数据接入服务负责对数据源平台的接入，可以提供满足对应平台安全机制的接入方式，并且支持扩充。

```
package crawler.api.service.oauthAccess;
import org.scribe.model.*;
public interface OAuthAccessService extend AccessService{
void fetchToken();
Token getToken();
}
```

如以上代码所示，其中 AccessService 是底层的数据接入服务接口，该服务是使用了 OAuth 2.0 的授权机制，即获取令牌 Token，因此需要实现 fetchToken 方法来向数据源平台请求接入的权限，将获取后的令牌封装在该类内部，需要使用授权时，可以通过 getToken 来重新获得令牌。通过这种方式，就可以把数据接入抽象为简单的服务接口，以适应更多其他的数据源接入方式。

（2）数据抓取逻辑

数据抓取服务负责对异构数据的抓取，其组织关系可以视具体的数据源平台中的信息组织方式以及具体的数据结构做出灵活更改。这部分接口实现逻辑灵活，但是最终输出必须满足上述的中间件规定才能与其他模块交互。

```
Public interface FetchService extends Runnable{
```

```
public void fetch();
public void init();
public void stop();
}
```

如以上代码所示，其中 FetchService 是底层的数据采集抽象服务接口，init 和 stop 方法分别用于起止操作时的准备工作，而数据抽取之前，必须通过 init 获取所需的数据平台接入安全信息，如果需要审核授权的即为用户提供授权流程。类中声明了 fetch 方法，用于存放抓取逻辑，具体的抓取逻辑可视情况而定，但必须保证其包含了对服务开启和关闭的准备工作，满足该接口的方法实现，才能接入到数据采集平台中，使用相关的数据采集资源，例如数据平台接入安全机制、元数据结构等。

3. 存储架构设计

系统的存储架构在设计时，由于需要考虑到不同的存储需求，以及后续的分析，首先应当把规范和实现分离。为了抽象数据存储的模块接口，我们把存储架构也分为两个部分，其一为数据库接入接口，其二为数据存储操作。

（1）数据库接入接口

数据库接入服务负责对数据库平台的接入，可以提供满足对应平台安全机制的接入方式，并且支持扩展。因此用户可以自由选择使用所需要的数据库来满足后续对数据分析处理的个性化需求。

（2）数据存储操作

数据库存储服务负责对已解析后的数据进行具体的存储操作，由于数据抽取模块的中间件规约，这部分存储逻辑相对简单，既可逐条存储抽取获得的数据，也可以设置缓冲区，将存储和抽取异步分离。

```
Public interface FetchService extends Runnable{
public void log(String uid,ArrayList<Weibo> weibo);
}
```

如以上代码所示，其中 FetchService 是底层的数据采集抽象服务接口，声明了 log 方法，用于存放、存储操作，此处的存储操作，将序列化的时间线，即一个微博列表存储到相应的用户 id 下，来表示一个用户与 feed 的关系，满足该接口的方法实现，才能接入到数据采集平台中，使用相关的数据采集资源，例如数据存储客户端、数据库检索服务等。

4. 数据库设计

数据库为了适应需求，需要对中断操作前的状态进行存储，以便于重新部署时不用再从头开始抓取，对每个节点操作也需要加入时间戳等标识信息，并且需要考虑到重复冗余数据的反复存取带来的性能降低。

本平台采用基于 key-value 的 NoSQL 数据库，在实现中使用了 Redis 内存数据库作为缓存。Redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string（字符串）、list（链表）、set（集合）、zset（sorted set，有序集合）和 hashset（哈希类型）。这些数据类型都支持 push/pop、add/remove、取交集、并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，Redis 支持各种不同方式的排序。与 Memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 Redis 会周期性地把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave（主从）同步。

如图 6.10 所示为采集平台数据关系图。我们将之前抽象的 Weibo 元数据信息存储于单个表中，按照不同的平台区分用户 User 的个人信息，并通过 feed_list 与 timeline 来关联两者之间的映射，这样可以有效减少数据冗余。

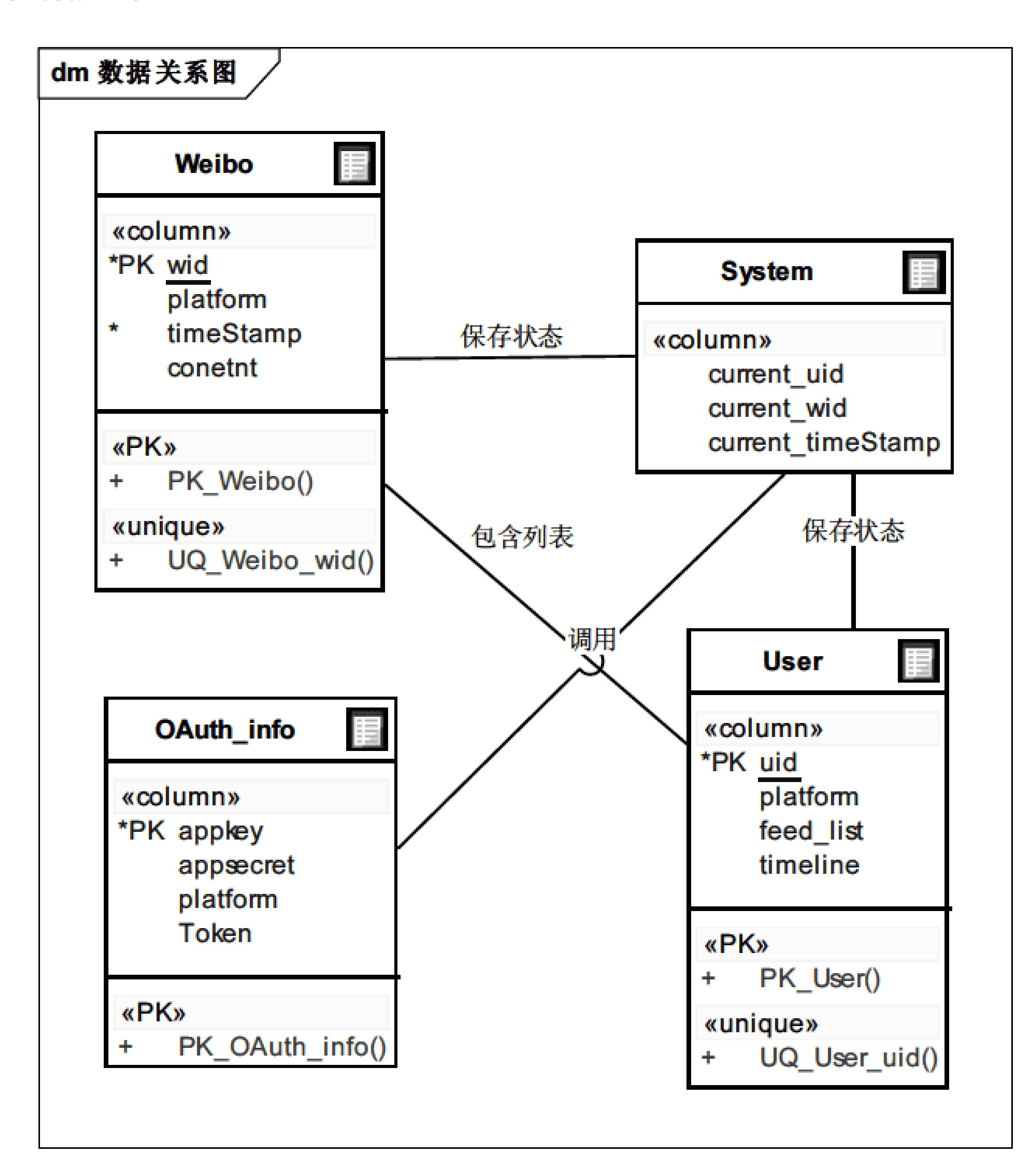


图 6.10　采集平台数据关系图

同时，为了提高抽取效率，增强可用性和系统错误恢复的能力，我们需要一个表单保存当前抽取的信息，如当前节点 User 的 uid、当前抽取 Weibo 的 wid。并且为了减少重复获取数据平台授权带来的安全隐患，以及减少使用复杂程度，可以将数据平台的申请授权信息、权限令牌等加密后保存于数据库中，方便日后调用。

建立好关系型数据后，可以将其用 key-value 的形式，在内存数据库中构建可分布式部署的数据，以满足高速存取与检索的需求。

6.4.2 数据采集插件的设计与实现

数据抽取、解析、存储模块插件，是本章系统的核心功能模块，且是唯一与外部异构数据接触的部分。

基本的抽取、解析、存储功能模块序列如图 6.11 所示。

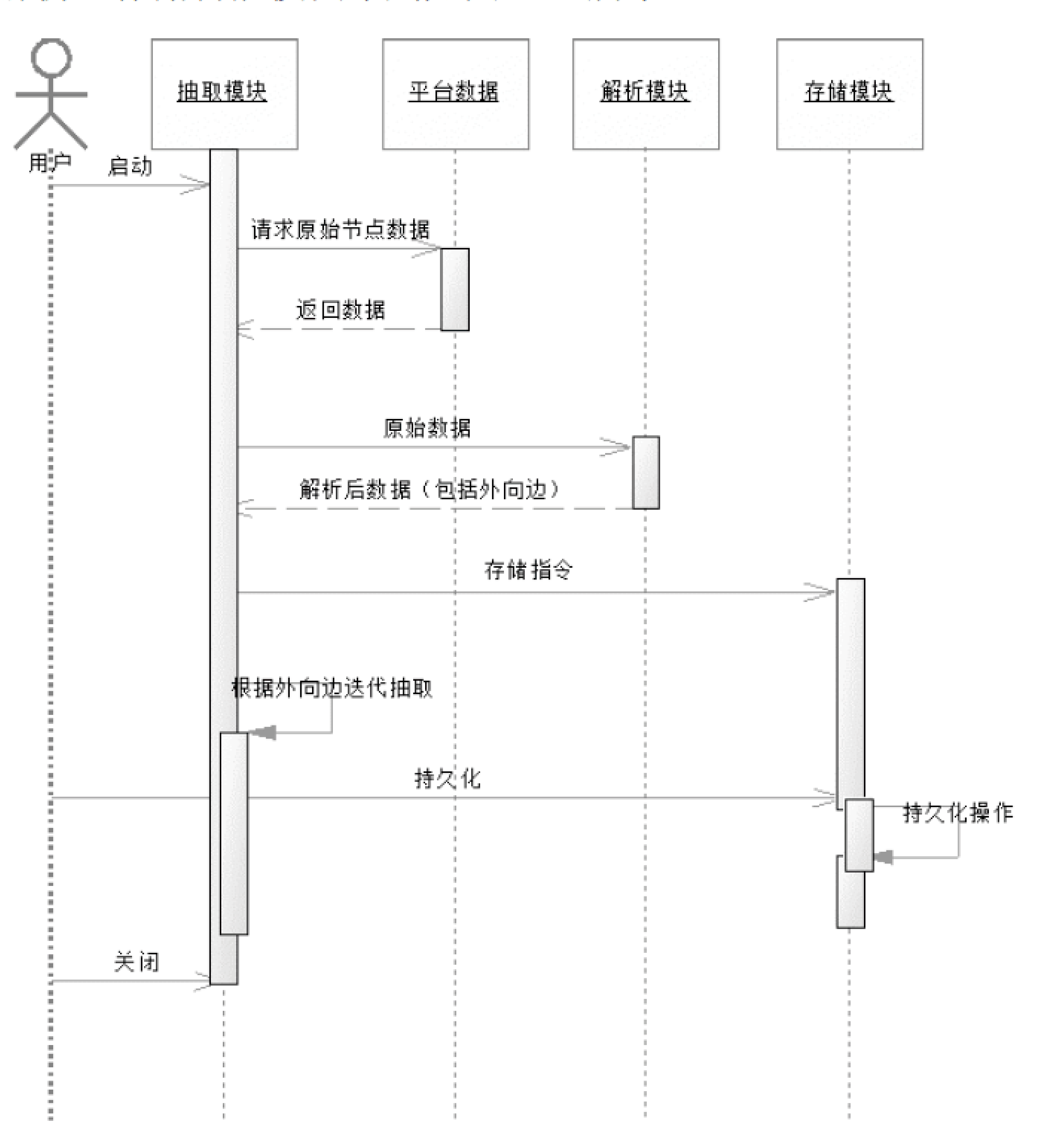


图 6.11 抽取、解析、存储功能模块序列图

以 Twitter 抽取解析存储插件的实现为例，该抽取插件的数据接入实现了 OAuthBundle 中声明的 OauthAccessService 接口。该接口使用了 Scribe-Java 实现的 OAuth 2.0 授权机制代码库，使用 HTTPClient 实现了 Twitter 的 RESTful API 数据请求。获得 JSON 数据，可用于后续的解析模块。每次抽取以用户 uid 为节点，好友 List 为外部链接，广度优先遍历以实现爬虫逻辑。在爬取的过程中，记录路径节点的数据，将其存储于内存数据库，等待持久化。

该插件的类图结构如图 6.12 所示。

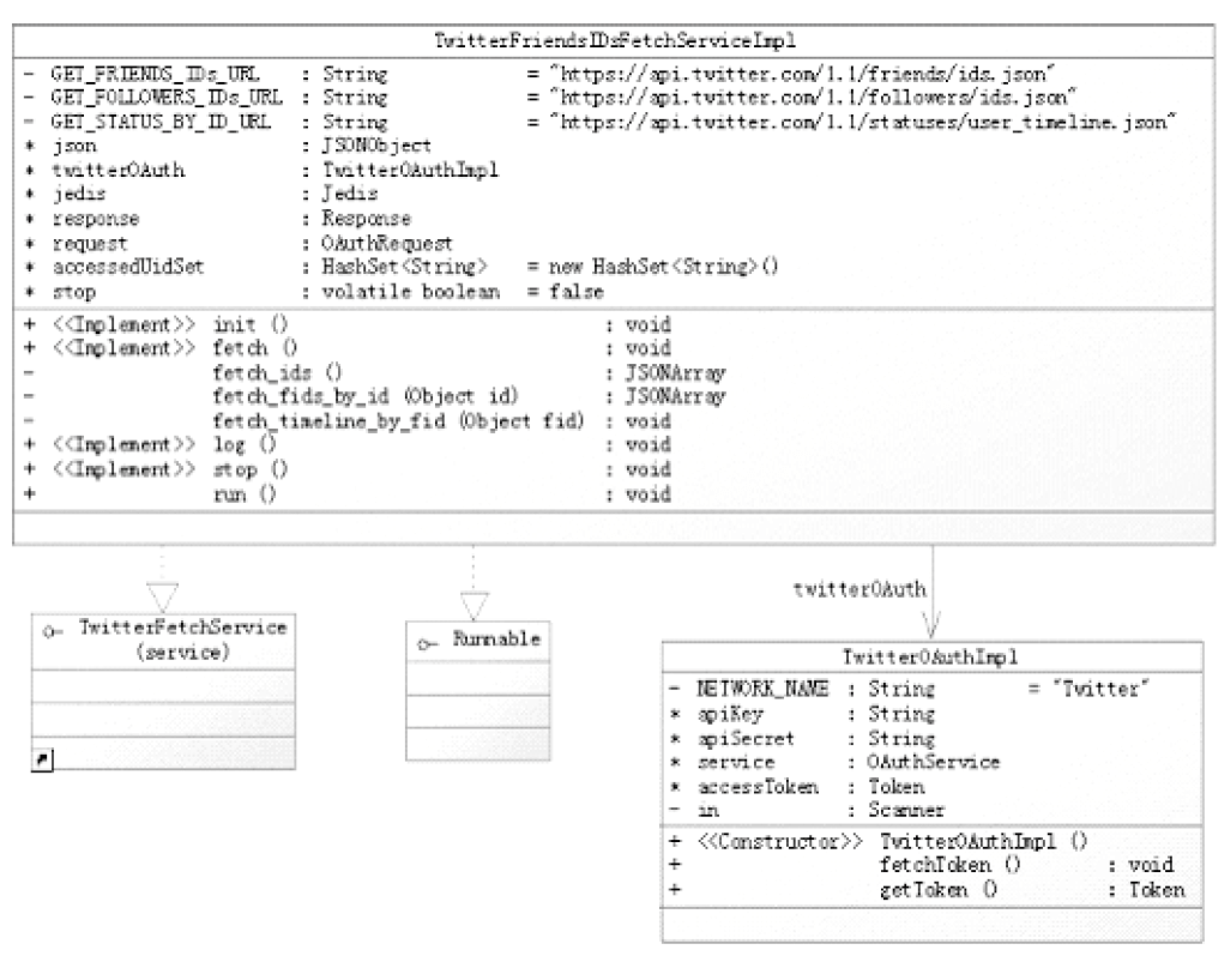


图 6.12　抽取、解析、存储功能插件类图

twitterFetchAPIBundle 的自动化构建脚本展示了相关依赖和 OSGi 所使用的依赖以及元数据信息，如下所示。

Gradle 自动构建脚本编译部分：

```
project(':twitterFetchAPIBundle')
    repositories {
        mavenCentral()
    }
    dependencies {
        compile 'net.sf.json-lib:json-lib:2.4:jdk15'
        compile project(':ScribeOAuthAPIBundle')
        compile project(':OAuthAPIBundle')
        compile project(':FetchAPIBundle')
        compile project(':JedisBundle')
    }
```

Gradle 自动构建脚本元数据定义：

```
    jar {
```

```
        manifest {
            version = '0.1'
            name = 'twitterFetchAPIBundle'
            symbolicName = 'twitterFetchAPIBundle'
            jar.archiveName = "twitterFetchAPIBundle.jar"
            instruction 'Bundle-Activator', 'crawler.api.fetch.Activator'
            instruction 'Import-Package' ,
                    'crawler.api.service.oauthAccess',
                    'crawler.api.service.fetch',
                    'org.osgi.framework;version="1.7.0"',
                    'org.osgi.util.tracker;version="1.5.1"'
            instruction 'Require-Bundle' ,
                    'ScribeOAuthAPIBundle',
                    'JedisBundle',
                    'JSON-libBundle'
        }
    }
```

每个 FetchServiceBundle 与之类似，数据的抽取依赖于特定的安全、接入的实现（crawler.api.service.oauthAccess，此处依赖于 ScribeOAuthAPIBundle 中的具体实现）和抽取逻辑接口（crawler.api.service.fetch，此处为该模块内部针对平台特征定义）。数据的解析依赖于特定的解析插件（JSON-libBundle）。数据的存储也依赖于特定的存储插件（JedisBundle）。

1. 抽取模块的设计与实现

（1）数据接入与抽取逻辑设计

数据接入的异构数据平台的特点之一就是数据接入方式的差异性。

针对这一点，我们把一些需要预先准备的授权机制，都包含在 OAuthBundle 中，通过声明 OAuthAccessService 接口来抽象数据接入的方法。

而抽取逻辑，按照原始数据的组织方式不同，按照拓扑结构来探索外部数据，将这部分逻辑抽象为包含 Fetch()方法的 FetchService 接口，可以通过迭代每次的外部链接，或者好友 ID 列表等方式来进行指定层数的广度优先遍历，以此实现数据采集的爬虫逻辑。

数据接入序列图如图 6.13 所示。

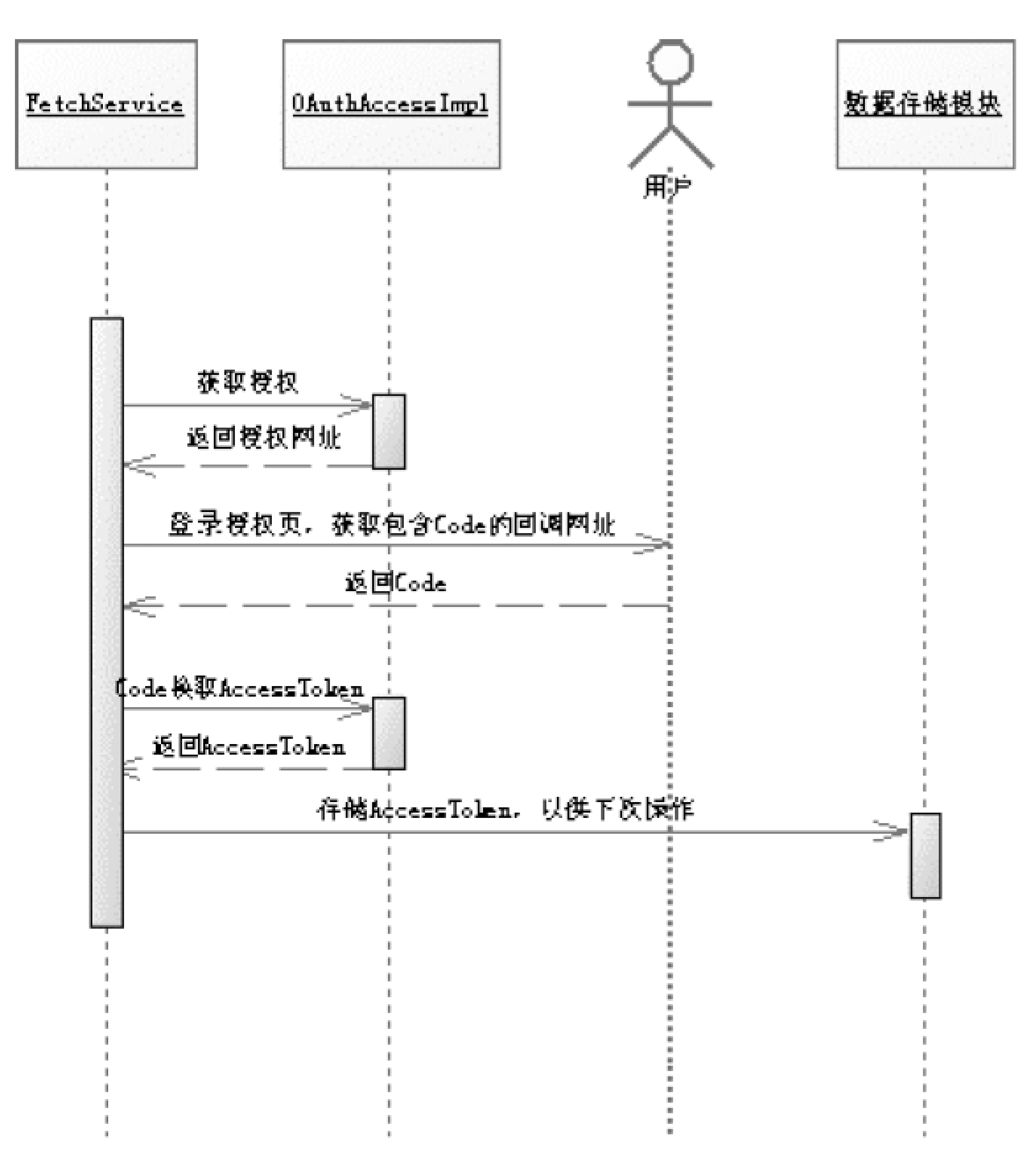


图 6.13　数据接入序列图

每次抽取数据前先校验数据库中的授权信息是否存在，如果不存在则请求授权，经过 OAuth 2.0 机制完成对数据平台的接入，并存储相关信息以供之后使用。

抽取逻辑序列图如图 6.14 所示。

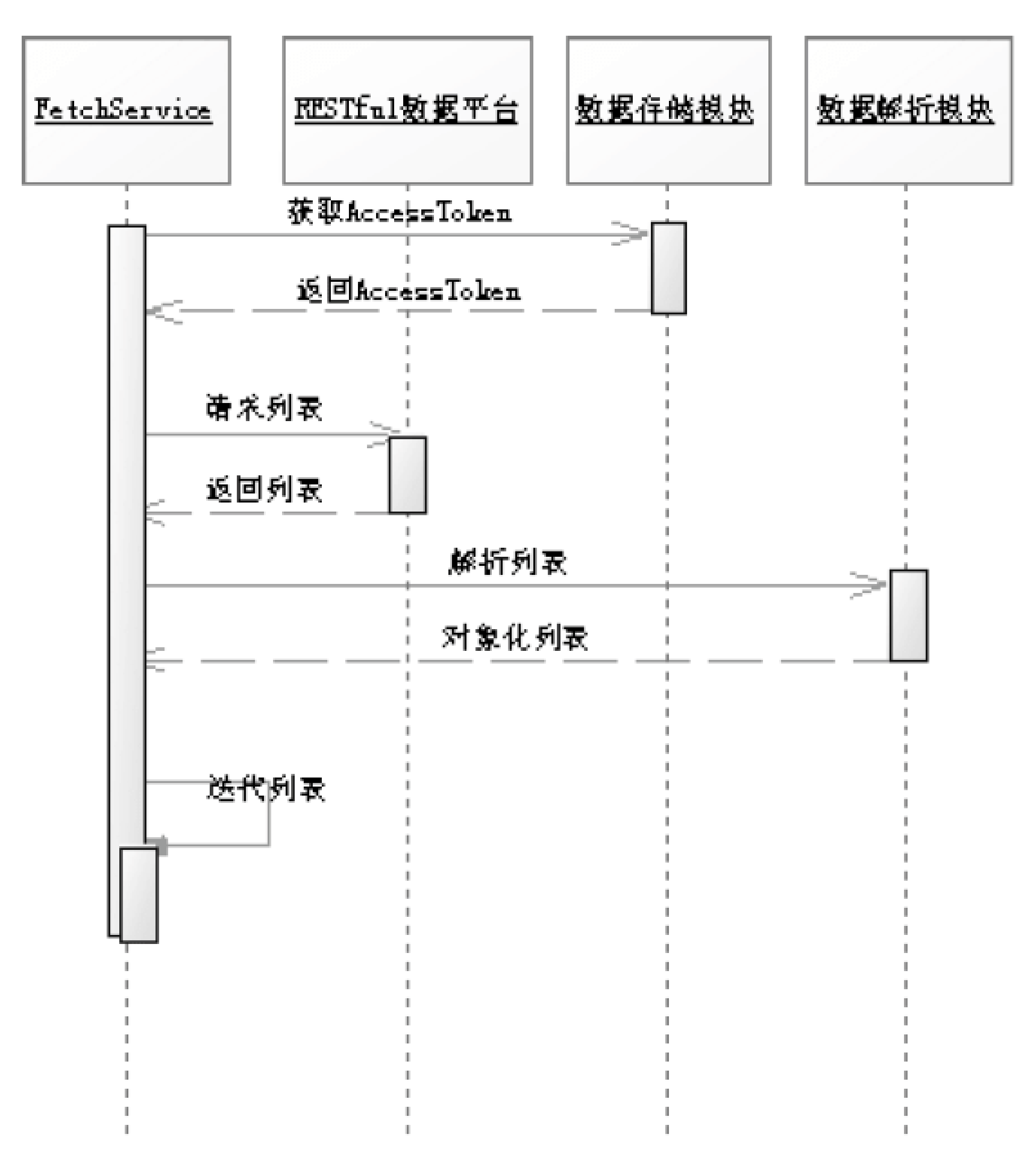


图 6.14　抽取逻辑序列图

抽取逻辑，即传统的网页爬虫逻辑，结合 SNS 网状结构，使用好友 ID 列表等信息代替外向

链接。将该列表序列化后，为 HTTP 请求提供参数，通过迭代抽取该列表数据即可实现深度优先的爬虫逻辑。

（2）抽取模块的实现

数据模块需要实现数据接入和抽取逻辑两个功能点，对应多平台数据，需要的接入方式也不尽相同，因此对于这个模块的接口抽象，首先需要考虑接入数据的方式，再解决抽取步骤迭代的逻辑部分。本章所述系统定义了 FetchService 服务接口，凡是实现该接口的类，均可作为插件接入系统的后续功能模块，FetchService 依赖于 OAuthBundle 内定义的授权方式，以及具体实现的 Fetch()方法定义的抽取逻辑。两者结合，组成了数据抽取模块。

以腾讯微博的 OAuth 2.0 实现为例，OAuth 允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的网站（例如，视频编辑网站）在特定的时段（例如，接下来的 2 小时）内访问特定的资源（例如仅仅是某一相册中的视频）。这样，OAuth 允许用户授权第三方网站访问他们存储在另外的服务提供者上的信息，而不需要分享他们的访问许可或他们数据的所有内容。

每个 FetchServiceImpl 类实现了继承自 FetchService 的具体服务接口，同时，为了并发特性，也实现 Runable 接口来实现多线程并发抽取，通过这种方式，可以有效利用网络延迟的时间来解析数据，同时也可以并行多个平台的抽取插件，以保证数据的实时性。

在获得接入权限后，利用获取的令牌，可以调用数据源平台 API 资源。

```
  public class TencentWeiboApi extends DefaultApi20 {
     private static final String AUTHORIZE_URL =
           "https://open.t.qq.com/cgi-bin/oauth2/authorize?client_id=
                  %s&response_type=code&redirect_uri=%s";

     private static final String SCOPED_AUTHORIZE_URL =
                  AUTHORIZE_URL + "&scope=%s";
     @Override
     public Verb getAccessTokenVerb() {
        return Verb.POST;
     }
     @Override
     public AccessTokenExtractor getAccessTokenExtractor() {
        return new TokenExtractor20Impl();
     }
     @Override
     public String getAccessTokenEndpoint() {
        return "https://open.t.qq.com/cgi-bin/oauth2/access_token?grant_type=
authorization_code";
     }
```

```
    @Override
    public String getAuthorizationUrl(OAuthConfig config) {
        // Append scope if present
        if (config.hasScope()) {
            return String.format(SCOPED_AUTHORIZE_URL,
                            config.getApiKey(),
                            OAuthEncoder.encode(config.getCallback()),
                            OAuthEncoder.encode(config.getScope()));
        } else {
            return String.format(AUTHORIZE_URL,
                            config.getApiKey(),
                            OAuthEncoder.encode(config.getCallback()));
      }
   }
}
```

如上面代码清单中所示，其中腾讯微博平台的数据采用 REST 方式组织，按照腾讯微博开放平台官方文档说明，可以通过 HTTP 客户端，请求数据源返回 JSON 格式的数据，通过用户的好友列表，可以构建关系网络，并通过迭代好友列表的方式，实现深度优先的数据抽取。

```
public void fetch() {
     for (Object data : fetch_datas()) {
        if (stop) return;
        for (Object fdata : fetch_fdatas_by_data((JSONObject) data)) {
           if (stop) return;
           log(((JSONObject) fdata).getString("openid"),
           fetch_timeline_by_fdata((JSONObject) fdata));
        }
     }
}
```

fetch()方法提供了数据的抽取逻辑，这里使用深度优先遍历，利用好友关系，构建网状结构的关系数据，并且通过解析重构节点数据，确保传入数据存储模块时的信息满足元数据模型规约的数据结构要求。

```
private JSONArray fetch_fdatas_by_data(JSONObject data) {
try {
         Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
```

```
    }
    String id = data.get("openid").toString();
    System.out.print("Now crawler at the id : ");
    System.out.println(id);
    JSONArray fdatas = new JSONArray();
    if (jedis.hget("tencent:uid:" + id, "followers_ids") != null)
        fdatas = JSONArray.fromObject(jedis.hget("tencent:uid:" + id,"followers_ids"));
    else {
        getJsonArray("0", id, fdatas);
        jedis.hset("tencent:uid:" + id, "followers_ids", fdatas.toString());
        }
    return fdatas;
    }
```

其中的迭代参数，是一个关注者 id 名单列表，通过一个 JSONArray 作为中间件来传递，并且使用解析器提取了键为 openid 的内容。之后的目标数据抽取，也是通过之前抽取的关注者 id 名单列表，对抽取后数据进行解析和重构，这部分工作交由解析器和抽取逻辑来协同完成，如下面的代码清单所示。

```
    private ArrayList<Weibo> fetch_timeline_by_fdata(JSONObject fdata) {
    if (stop) return new ArrayList<Weibo>();
    String fid = fdata.get("openid").toString();
    if (accessedUidSet.contains(fid))return new ArrayList<Weibo>();
        try {
       Thread.sleep(5000);
    } catch (InterruptedException e) {
       e.printStackTrace();
    }
    System.out.print("Now crawler at the id : ");
    System.out.println(fid);
    request = new OAuthRequest(Verb.GET,GET_STATUS_BY_ID_URL);
    tencentOAuthImpl.service.signRequest(tencentOAuthImpl.accessToken, request);
    request.addQuerystringParameter("format", "json");
    request.addQuerystringParameter("fopenid", fid);
    request.addQuerystringParameter("oauth_consumer_key", tencentOAuthImpl.apiKey);
    request.addQuerystringParameter("openid", tencentOAuthImpl.client_id);
    request.addQuerystringParameter("oauth_version", "2.a");
    request.getCompleteUrl();
    response = request.send();
```

```
    json = JSONObject.fromObject(response.getBody());
    accessedUidSet.add(fid);
    ArrayList<Weibo> timeline = new ArrayList<Weibo>();
    if (json.containsKey("data") && json.getJSONObject("data").containsKey("info"))
       for (Object object : json.getJSONObject("data").getJSONArray("info")) {
          JSONObject jsonObject = (JSONObject) object;
           if (jsonObject.containsKey("id") && jsonObject.containsKey("id")) {
           timeline.add(new Weibo(jsonObject.getString("id"), jsonObject.
getString("text")));
             }
    }
    return timeline;
    }
```

2. 解析模块的设计与实现

数据解析是将异构的原始数据提取出关键的元数据信息，其中包括节点的拓扑结构，以及节点本身的数据，我们对数据的基本模型抽象后得出一个通用的存储格式，而数据解析正是从多平台异构数据中取得这些共性内容，再组织后存储，以便后续的数据挖掘、分析研究。

（1）解析模块的设计

解析模块的序列图如图 6.15 所示。

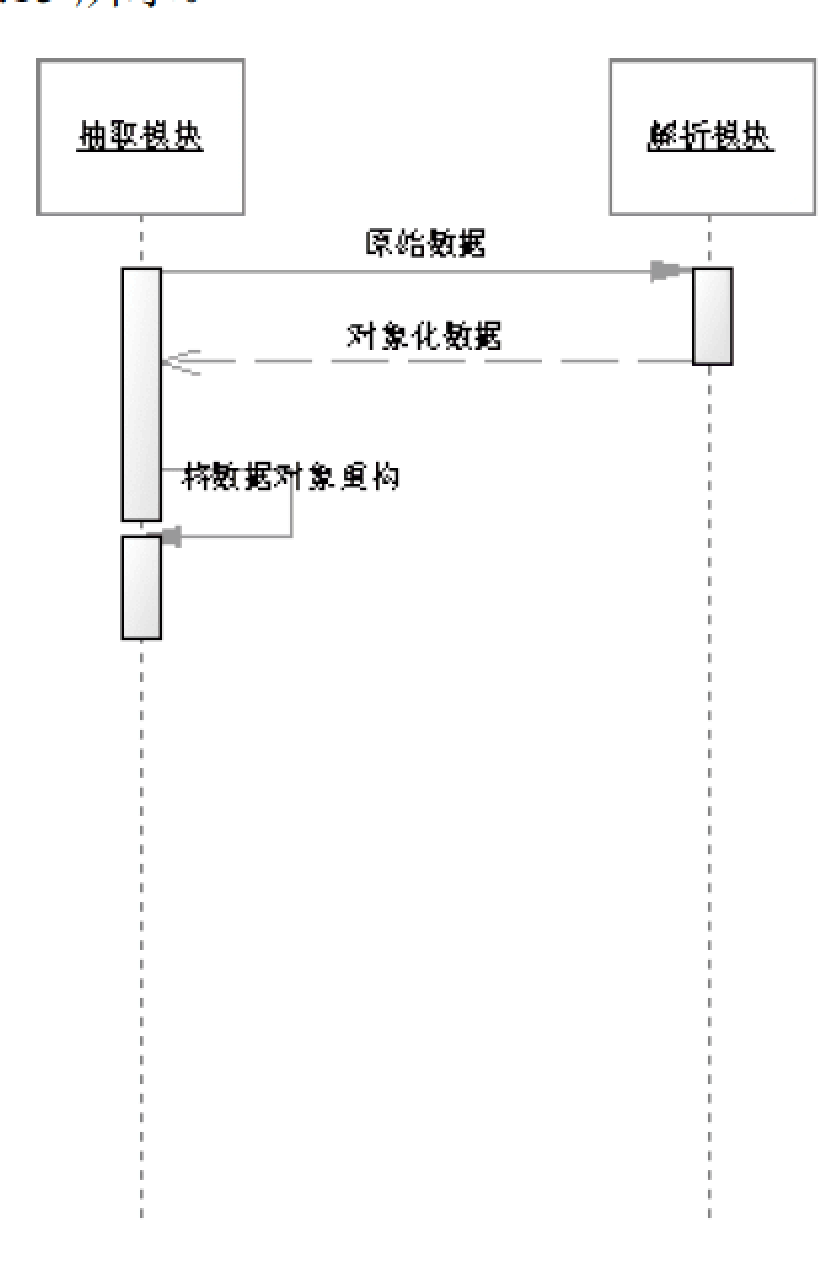


图 6.15　解析模块序列图

解析模块相对其他模块使用更频繁，在设计时更多考虑了高内聚低耦合的要求，对数据的传

输仅仅依靠纯文本来作为参数和返回值。这样有助于解析过程独立于系统存在，也方便抽象出接口，以供实现时的代码更加简洁明了。

（2）解析模块的实现

对数据的解析和重构是本系统插件主要解决的问题，在插件所实现的抽取和存储逻辑中均需要使用解析服务，为此可以直接引用现有的工具库，例如由于 Twitter 使用的数据格式采用了 JSON，因此调用了 JSON 的库。于是在 OSGi 的头文件中声明 JSON-lib 中相应的包名作为解析所需的 Import-Package。

对于其他的需求，同样可以使用相应实现的解析包来协助我们把数据重构成统一的格式。其方法与本例相同，也是将解析服务封装成 Bundle 后，在头文件中声明 import 包名。在此不再赘述。

3. 存储模块的设计与实现

为了异构数据的高效率抽取，并且方便之后的使用、分析，针对单一平台、单一数据库模式带来的低效率以及受磁盘 IO 等环境影响的问题，本系统支持分布式的数据抽取和存储方式。对于存储模块，本系统使用了 Redis 内存数据库中间层，以作为持久化数据库的补充，其实现使得内存的高效与磁盘的低成本大数据存放。

（1）存储模块的设计

该模块的序列图如图 6.16 所示。

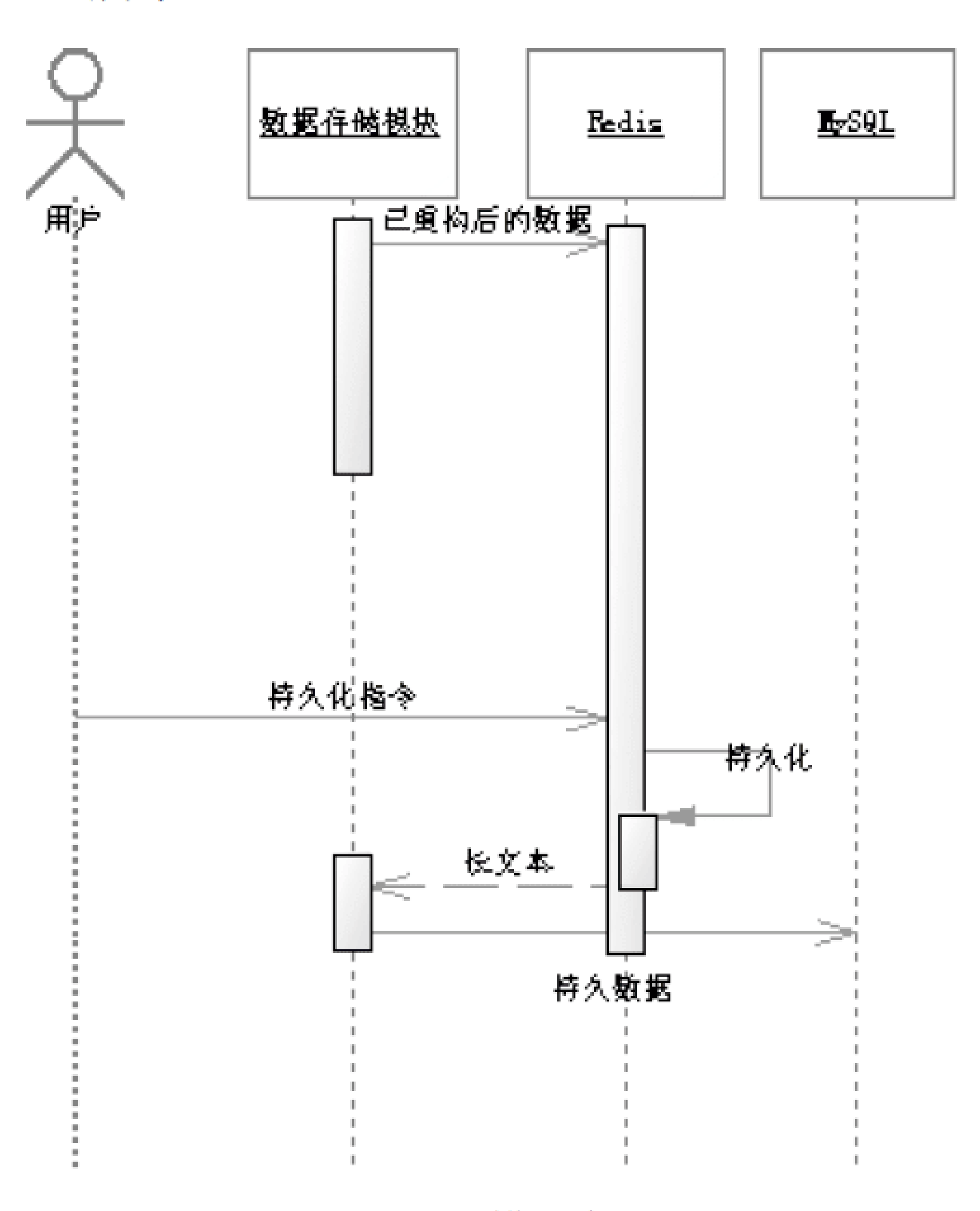


图 6.16　存储模块序列图

本系统的数据存储考虑到大数据量、不断变化的需求都会遇到磁盘 IO 效率受限的问题，由此会影响到插件的抽取效率，因此设计时采用了 Redis 的内存数据库，作为中间层缓存，这样在

需要快速存取数据时可以发挥内存数据库高速搜索的优势，而当遇到需要持久化时，在服务上发送指令即可完成由 Redis 向 MySQL 的 Dump 操作。

（2）存储模块的实现

数据存储的过程中，需要考虑写入和覆写带来的磁盘寿命的问题，本系统中采用了 COW（Copy on Write）机制以及内存数据库来简化这一问题。

如下面的代码清单所示，对腾讯微博数据的存储可以简单抽象为将每个用户节点与其时间线数据映射的存储。

```
public void log(String uid, ArrayList<Weibo> timeline) {
jedis.hset("tencent:uid:" + uid, "time_line", JSONArray.fromObject(timeline).
toString());
}
```

而数据节点间的组织关系，则由抽取时获得的用户关注列表来表示，且通过检验的手段避免重复写入列表。

```
if (jedis.hget("tencent:uid:" + id, "followers_ids") != null)
    fdatas = JSONArray.fromObject(jedis.hget("tencent:uid:" + id, "followers_ids"));
else {
     getJsonArray("0", id, fdatas);
     jedis.hset("tencent:uid:" + id, "followers_ids", fdatas.toString());
  }
```

与数据抽取模块相似，在数据存储模块中要处理的内容是数据抽取模块与数据解析模块协同合作完成的。当数据需要更新或添加时，首先使用 Jedis 客户端来对 Redis 内存数据库缓存，而当需要持久化时，再调用相关函数，提取内存中的数据，保存到磁盘上。

6.4.3　系统服务框架的设计与实现

系统服务框架是本系统的管理模块，是负责管理各数据抽取、解析、存储插件模块生命周期以及提供添加更新服务的框架。

该模块需要实现 OSGi 规范约定，以 Bundle 机制提供初始化和结束时的具体处理，为了适应平台的增加、内容的变化，因此要把该模块的接口与具体插件实现分离，单纯地作为组织管理的实现，为用户提供友好的后台管理界面，并且采用 RESTful 设计模式，将对框架的操作简化为状态指令，来适应今后对多平台控制方式的需求变化。

1. 服务框架控制插件的设计与实现

（1）插件起止操作的设计

遵循 OSGi 的规范，本系统中的插件服务，按照示例 Activator 的设计，需要实现两个方法，

用于为插件开启前准备和关闭前的状态保存等操作。

对数据的具体操作都封装在数据抽取、解析、存储插件中，因而服务框架只需要专注管理这些线程的调度问题即可。

插件起止操作的序列图如图 6.17 所示。

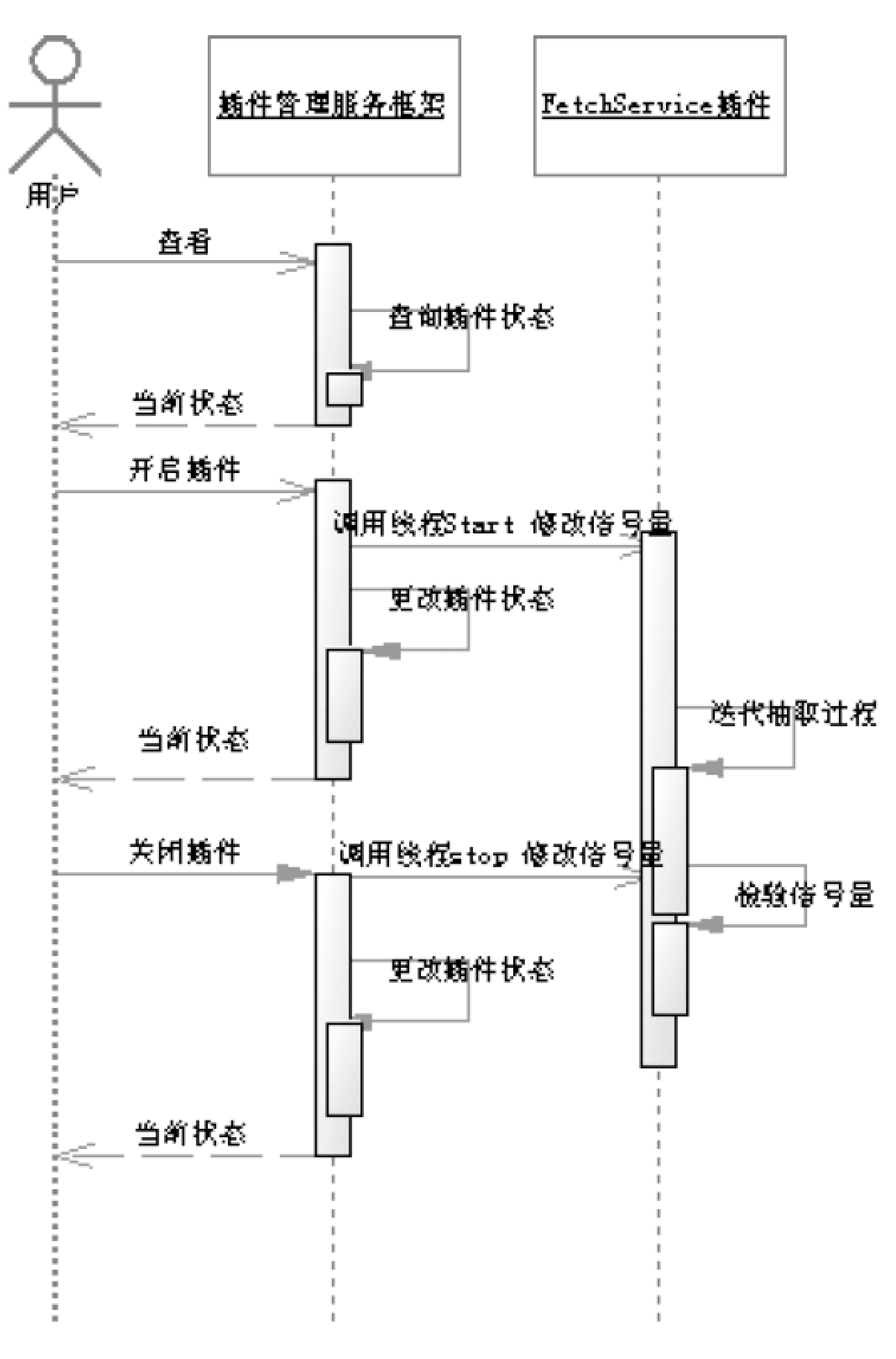


图 6.17　插件管理激活器序列图

在插件中实现的 Runable 接口，由于 Java 的线程结束机制，必须将占用资源返还 JVM 后再进行关闭线程的原子操作。为此，可以设置一个信号量 stop 用以表示该插件的状态，该信号量对外部可见。服务框架模块的插件起止操作对该信号量进行监控和修改，而插件内部每次抽取迭代都会对该信号量检验，当该信号量被更改，插件就会执行保留当前状态的操作，并停止工作进入阻塞态。

（2）插件起止操作的实现

根据 OSGi 规约，插件服务的启动与结束操作，可以抽象为 Activator 的 start()与 stop()接口实现，凡是实现该方法的插件，均可以接受 OSGi 框架的动态绑定管理，为了数据的安全，以及抽取的效率，本系统把该框架操作上升到线程级别，通过改变插件内部的线程信号量，来告知插件需要进行起止的准备工作，并且对框架管理服务模块反馈，以此告知用户是否起止完毕，并且便于后续的客户端随时监控各插件的状态。

该模块实现的类图如图 6.18 所示。

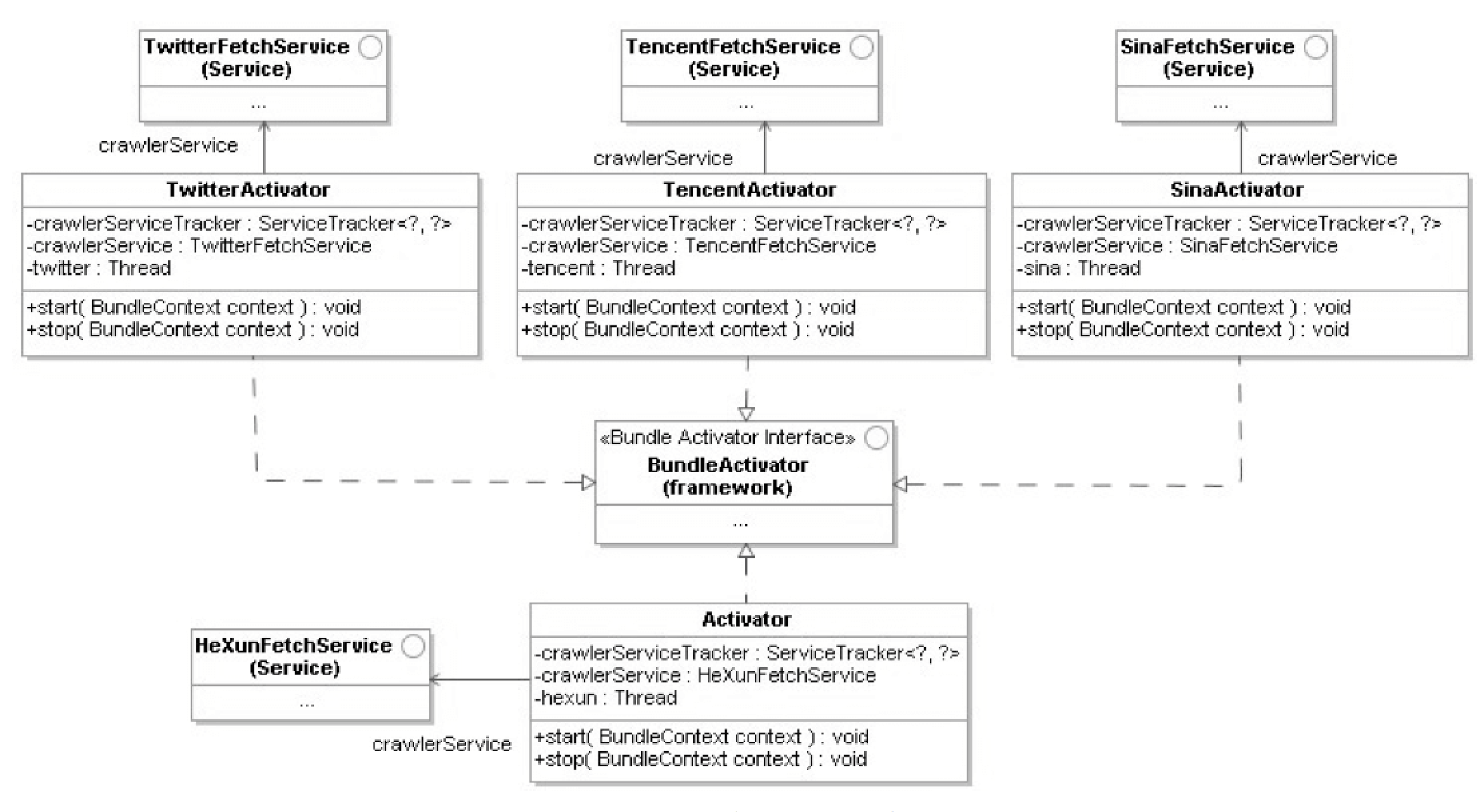


图 6.18　插件管理激活器类图

当 start()方法被调用时，信号量 stop 设为 false，则在插件的抽取模块中，可以迭代抽取逻辑部分，直到深度遍历结束。

当 stop()方法被调用时，信号量 stop 设为 true，则该插件的验证信号量被激发，迭代抽取逻辑部分进入阻塞态并释放所占用的资源，在这之前，对当前抽取节点的数据进行保存，并且等待下一次信号量改变。

2. 服务框架管理工具的设计与实现

根据框架管理功能需求的约定，插件服务框架需要实现一个基于 RESTful 的管理工具。

（1）管理工具的设计

该工具需要实现的功能包括：监视插件运行状态，管理员能够通过发送指令控制插件的开启和关闭，并能添加或更新插件。

插件起止操作的序列如图 6.19 所示。

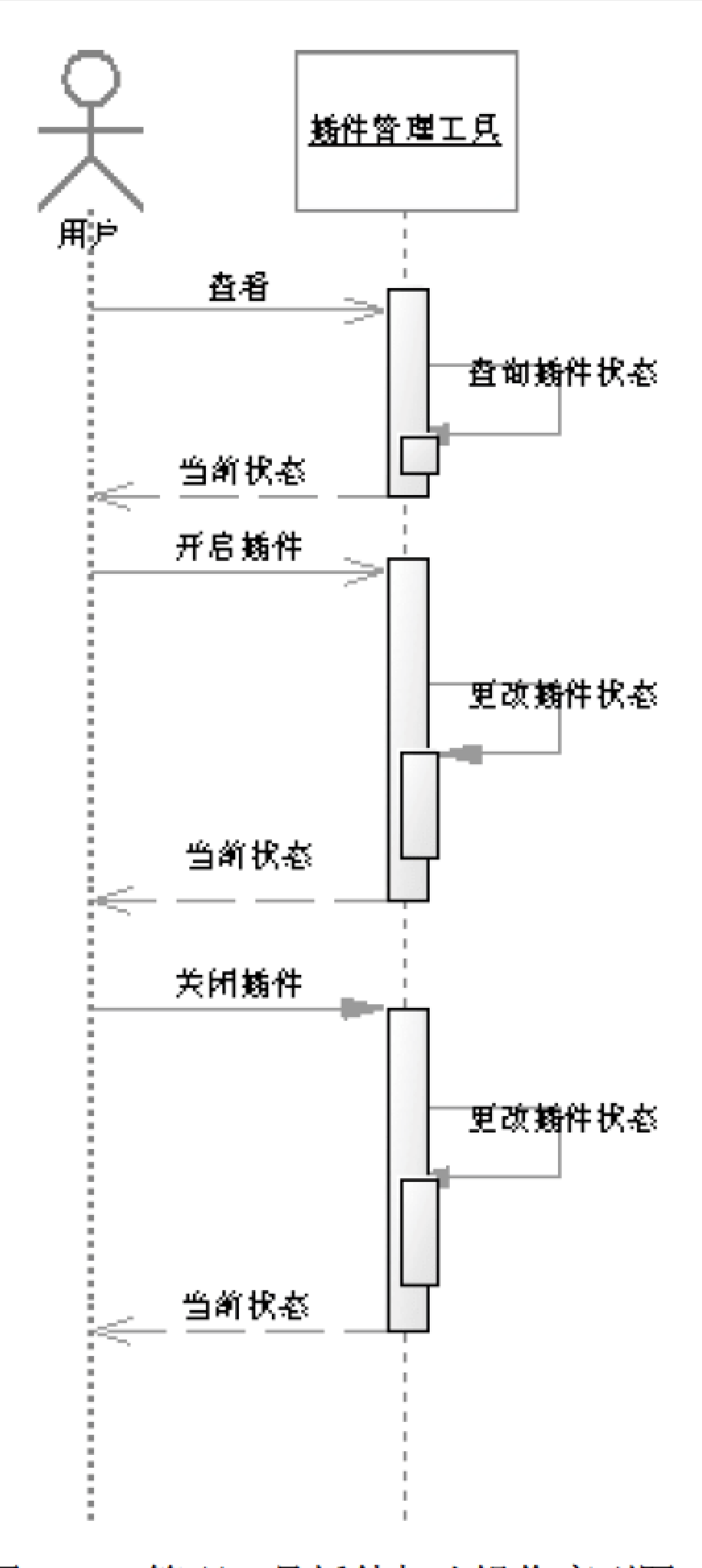
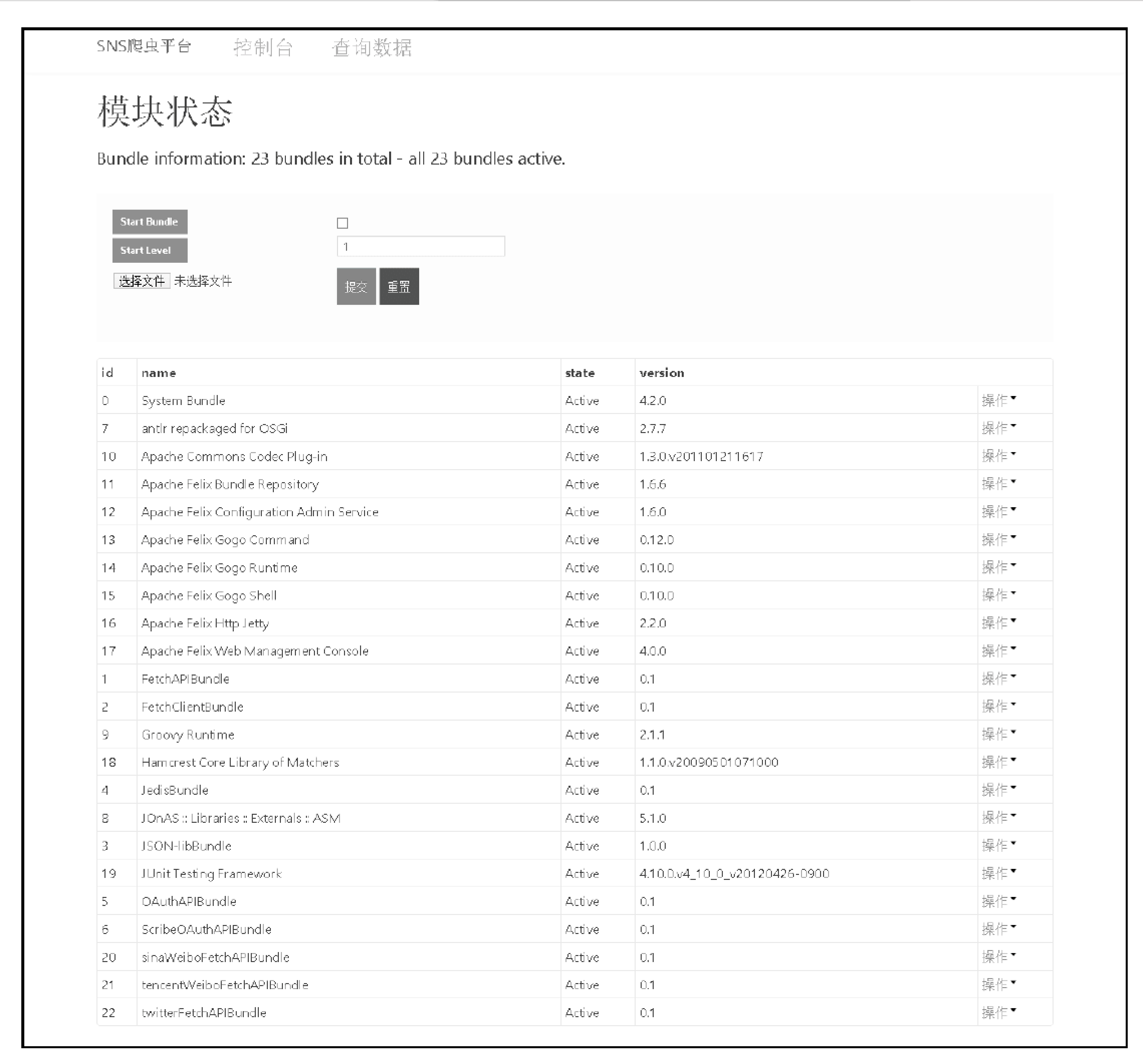


图 6.19　管理工具插件起止操作序列图

插件管理工具的功能用例主要包括三个内容，其中插件状态是跟踪 OSGi 框架内部实现的 Bundles 状态，而开启、关闭插件以及更新插件的操作均基于 OSGi 框架的 RESTful 特性，通过客户端的 POST 会话，上传新插件或更新插件。

每当操作完成，均会调用查询命令，以此更新插件的状态显示，保证监控的及时性。管理工具的控制台界面设计如图 6.20 所示。



id	name	state	version	
0	System Bundle	Active	4.2.0	操作
7	antlr repackaged for OSGi	Active	2.7.7	操作
10	Apache Commons Codec Plug-in	Active	1.3.0.v201101211617	操作
11	Apache Felix Bundle Repository	Active	1.6.6	操作
12	Apache Felix Configuration Admin Service	Active	1.6.0	操作
13	Apache Felix Gogo Command	Active	0.12.0	操作
14	Apache Felix Gogo Runtime	Active	0.10.0	操作
15	Apache Felix Gogo Shell	Active	0.10.0	操作
16	Apache Felix Http Jetty	Active	2.2.0	操作
17	Apache Felix Web Management Console	Active	4.0.0	操作
1	FetchAPIBundle	Active	0.1	操作
2	FetchClientBundle	Active	0.1	操作
9	Groovy Runtime	Active	2.1.1	操作
18	Hamcrest Core Library of Matchers	Active	1.1.0.v20090501071000	操作
4	JedisBundle	Active	0.1	操作
8	JOnAS :: Libraries :: Externals :: ASM	Active	5.1.0	操作
3	JSON-libBundle	Active	1.0.0	操作
19	JUnit Testing Framework	Active	4.10.0.v4_10_0_v20120426-0900	操作
5	OAuthAPIBundle	Active	0.1	操作
6	ScribeOAuthAPIBundle	Active	0.1	操作
20	sinaWeiboFetchAPIBundle	Active	0.1	操作
21	tencentWeiboFetchAPIBundle	Active	0.1	操作
22	twitterFetchAPIBundle	Active	0.1	操作

图 6.20　管理工具的控制台界面

（2）管理工具的实现

本管理工具是 B/S 架构的 Web 应用，基于 Node.js 的轻量 Web 框架 Express 开发，该框架是 MVC 架构，将 RESTful 的插件状态 json 数据放到 view 模板解析。而与框架交互的命令等需要考虑安全问题的功能，则在 Server 端执行。

本工具核心代码，是基于 URL 路由的命令，因此也满足 RESTful 设计规范，其实现的核心代码如下所示。

控制台 Bundles 状态获取代码：

```
app.get('/console',function(req,res){
var list='';
var options = {
  hostname:'felix.torchz.net',
  port:8080,
```

```
    path:'/system/console/Bundles.json',
    auth:':'
  };
  http.get(options, function(response) {
    response.setEncoding('utf8');
    response.on('data',function(chunk){
    list=list+chunk;
  });
  response.on('end',function(){
    res.render('index',{title:'console',Bundles:JSON.parse(list)});
  });
  });
```

控制台命令发送代码：

```
  app.get(/^\/console\/(\w+)$/,function(req,res){
  var url = '/system/console/Bundles/' + req.query.id;
  var data = querystring.stringify({action: req.params[0]});
  var options = {
    hostname:'felix.torchz.net',
    port:8080,
    path:url,
    auth:':',
    headers:{'Content-Type':'application/x-www-form-urlencoded'},
    method:'POST'
  };
  var request = http.request(options, function(response) {
    response.setEncoding('utf8');
  });
  request.write(data);
  request.getHeader('Content-Type');
  request.end();
  res.redirect('/console');
  });
  app.listen(3000);
```

由于本工具使用 B/S 架构，为了提高浏览器兼容性，并且避免不同前端 JavaScript 解释器实现不同可能造成的异常，将 URL 路由与服务管理命令相关联，只需要单击链接即可发送命令到框架，完成开启关闭插件的任务，并且支持上传插件文件，以此来调用框架动态绑定机制，实现动态更新插件内容。

6.5 部署与测试

6.5.1　系统部署

1. 数据采集平台的部署

首先确保系统包含 Java 1.6 以上的虚拟环境，并且包含 JDK 开发组件。本数据采集平台基于 OSGi 标准构建，因此支持 OSGi 的各个框架实现。在本节中以较为流行的 OSGi 实现框架 Felix 为例介绍整个系统的部署方法。

Felix 是一个 OSGi 版本 4 规范的 Apache 实现。

OSGi 是一个基于 Java 的服务平台规范，其目标是被需要长时间运行、动态更新、对运行环境破坏最小化的系统所使用。有许多公司（包括 Eclipse IDE，它是第一个采用 OSGi 技术的重要项目）已经使用 OSGi 去创建其微内核和插件架构，以允许在运行时刻获得好的模块化和动态组装特性。其他几个项目如 Apache Directory、Geronimo、Jackrabbit、Spring 以及 JOnAS 也都正在转向采用 OSGi。

目前 Felix 已经实现了 OSGi R4 规范中的大部分内容。Felix 的官方地址为：http://felix.apache.org/site/index.html。

在获取 Felix 最新版本的已编译文件后，可以通过$ java -jar bin/felix.jar 来启动框架。

数据存储模块使用了 Redis 数据库作为数据缓存的内容，因此，在部署环境时还需要编译并部署 Redis 内存数据库服务，以供数据抽取存储平台的客户端调用。

Redis 是一个高性能的 key-value 数据库。Redis 的出现在很大程度上补偿了 Memcached 这类 key/value 存储的不足，在部分场合可以对关系数据库起到很好的补充作用。它提供了多种语言编写的客户端，使用和扩展都十分方便。

在 Redis 官方网站 http://www.redis.io/下载 Redis 数据库源码，或通过以下方式来下载编译 Redis 数据库：

```
$ wget http://redis.googlecode.com/files/redis-2.6.14.tar.gz
$ tar xzf redis-2.6.14.tar.gz
$ cd redis-2.6.14
$ make
```

至此，Redis 数据库编译完成，运行$ src/redis-server，可以看到数据库服务器启动的欢迎界面。保持该 server 进程运行，可为 Redis 客户端提供内存数据管理服务。

另外，当需要进行分布式部署时，OSGi 框架也支持远程部署，并且可以通过自带的 Bundle-start-level 来确定部署各插件时的启动顺序和优先级。

在多台服务器的部署上，可以引用 Redis 数据库的主从分布式架构，也可以单独提取多个服

务器的数据，通过合并操作来完成对数据的分布式处理和统一持久化。

2. 数据采集平台管理工具的部署

数据采集平台的管理工具，独立于原 OSGi 框架，以便于对其部署时，不用考虑环境情况，由 Felix 自带的 OSGi Web Console 进行满足 RESTful 的指令交互。用户可以使用数据采集平台的管理工具来监控插件运行时信息，并且支持对插件的扩展和更新，这一系列操作独立于其他插件，可以动态部署，并支持安全机制的添加，管理员可以要求对管理工具的权限进行分配，要求用户登录后才能获得开关、上传、更新插件的权限。

数据采集平台的管理工具基于 Node.js 的异步编程环境，不管是部署环境还是对功能的扩展非常方便快捷，采用 B/S 架构，可以适应更多的平台接入该管理工具，同时也保证了服务随时可用，支持远程控制等额外需求。

Node.js 是一个可以快速构建网络服务及应用的平台。该平台的构建基于 Chrome's JavaScript Runtime，也就是说，实际上它是对 Google V8 引擎（应用于 Google Chrome 浏览器）进行了封装。

V8 引擎执行 JavaScript 的速度快，性能好。Node 对一些特殊用例进行了优化，提供了替代的 API，使得 V8 在非浏览器环境下运行得更好。例如，在服务器环境中，处理二进制数据通常是必不可少的，但 JavaScript 对此支持不足，因此，V8.Node 增加了 Buffer 类，方便并且高效地处理二进制数据。因此，Node 不仅仅简单地使用了 V8，还对其进行了优化，使之适应更多的服务端环境。

Node.js 作为一个新兴的后台语言，有很多吸引人的地方：RESTful API、单线程、非阻塞 IO、V8 虚拟机、事件驱动，Node.js 可以在不新增额外线程的情况下，依然可以对任务进行并行处理——Node.js 是单线程的，它通过事件轮询（Event Loop）来实现并行操作，对此，我们应该要充分利用这一点——尽可能地避免阻塞操作，多使用非阻塞操作。

部署 Node.js 也非常简单，以 Linux 的发行版 Ubuntu 为例，首先确保系统包含所有的包。

```
$ sudo apt-get install g++ curl libssl-dev apache2-utils
$ sudo apt-get install git-core
```

准备好系统环境后，在 Node.js 官网 http://nodejs.org/ 下载二进制已编译包，或者源码自行编译。设置好环境变量后，即可调用 node 解释器。

```
$ export PATH="$HOME/local/node/bin:$PATH"
$ export NODE_PATH="$HOME/local/node:$HOME/local/node/lib/node_modules"
```

在这里直接运行之前写好的脚本：

```
$ node console/app.js
```

之后通过浏览器访问 http://localhost:3000/console 即可见到如图 6.21 所示的界面。

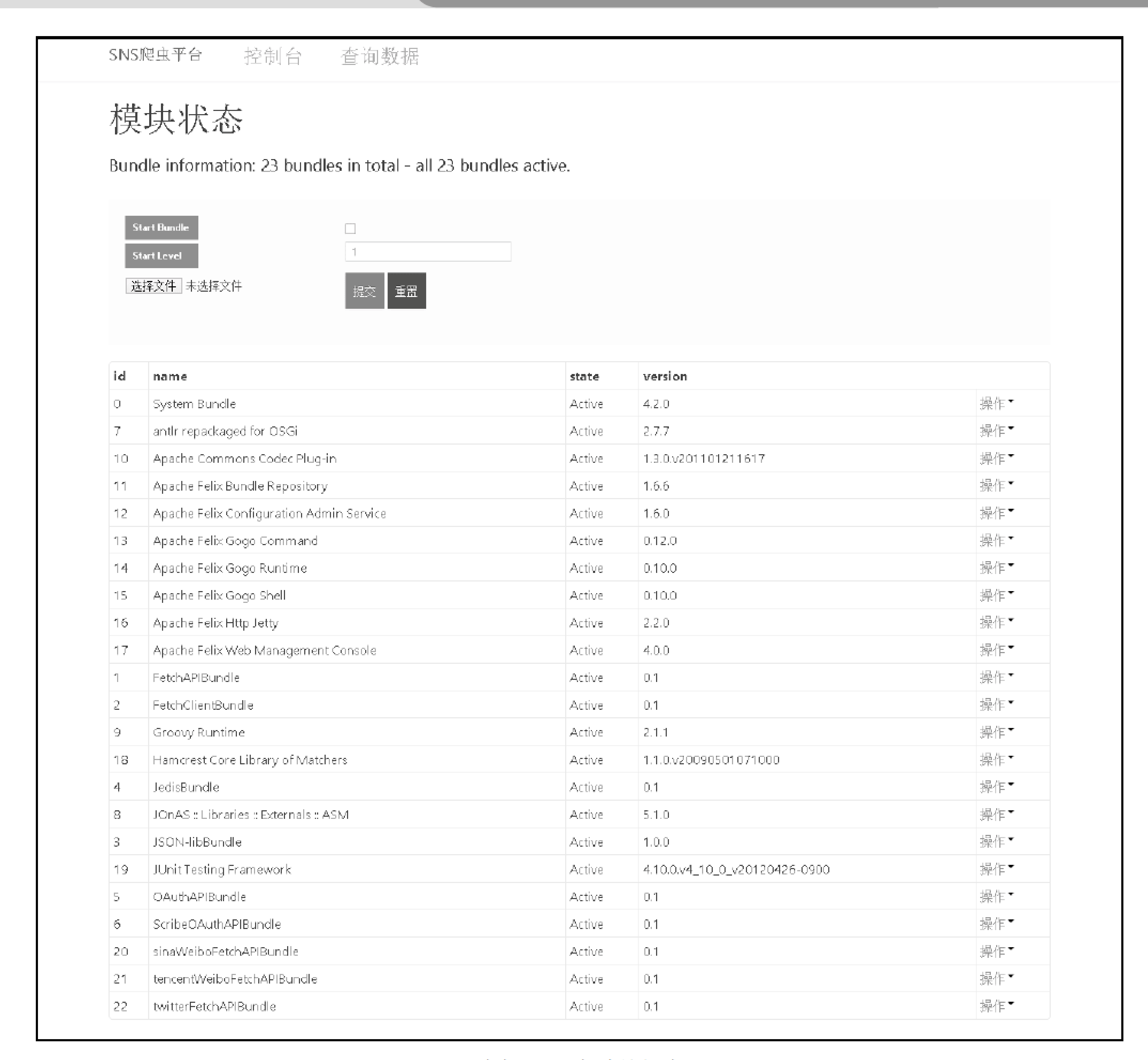


图 6.21　运行界面

在该界面中，可以支持对插件的更新、加载、启动、停止等操作，得益于分布式部署的情况，我们可以在不同的平台上访问这些数据采集平台的资源，并且能够及时做出修改。

6.5.2　系统测试

1. 功能测试结果与分析

本系统的控制台是需要进行功能测试的主要模块，围绕服务框架插件管理的各项功能，按照测试用例设计，对该模块进行了功能测试。

如图 6.22 为插件管理工具页面，包含了插件当前信息的展示。

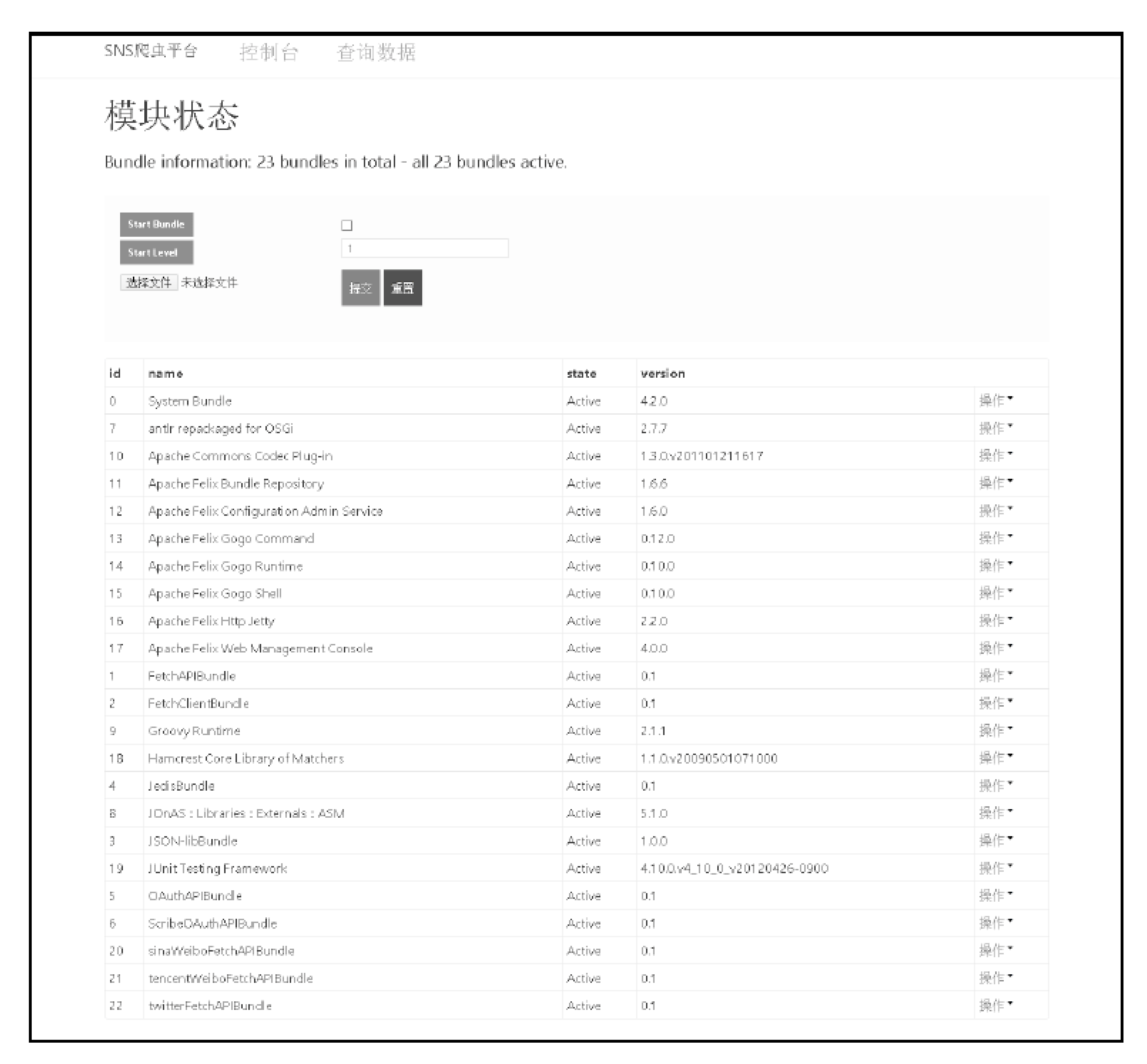


图 6.22　插件管理工具页面

系统的各模块插件状态信息都会展示在控制台页面，包括插件 id、插件名、插件所处状态、版本号等信息，并且可以通过单击操作指令，开启和关闭插件。

如图 6.23 所示为更新/添加插件功能测试。

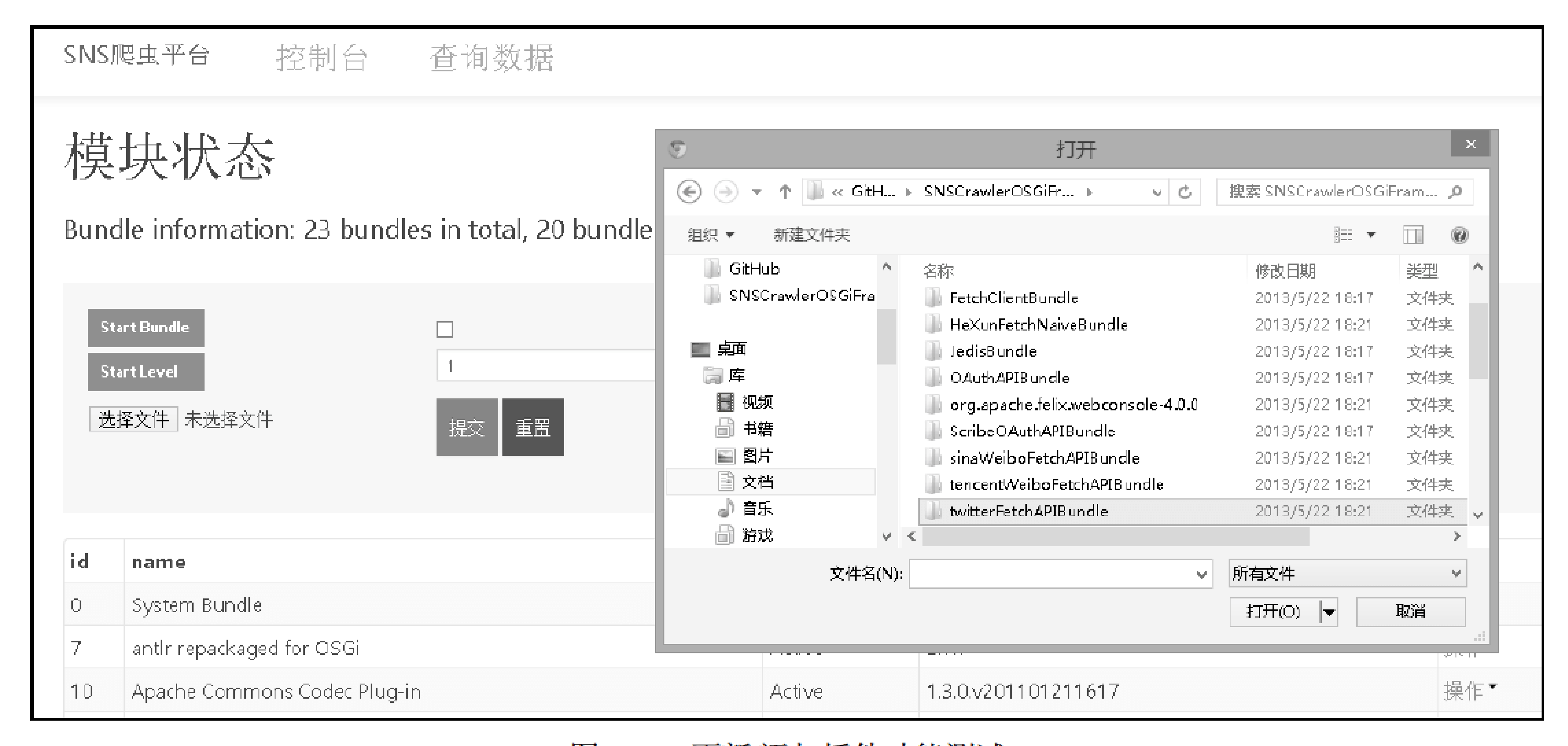


图 6.23　更新/添加插件功能测试

通过单击选择文件，设置相关选项，并提交新插件文件，可以更新插件版本，或增加新的插件服务。经测试新上传的插件可以经由 OSGi 动态绑定机制，良好部署并运行在平台中。

2. 性能测试结果与分析

性能测试环境如表 6.1 所示。

表 6.1　性能测试环境约定

<table>
<tr><td>软件语种</td><td colspan="2">中文、英文</td></tr>
<tr><td>测试时间</td><td colspan="2">2013 年 05 月 15 日至 2013 年 05 月 20 日</td></tr>
<tr><td rowspan="2">测试环境</td><td>硬件</td><td>型号：客户机 i7/4GB
服务器：Xeron 1.6GHz/1024MB CentOS</td></tr>
<tr><td>软件</td><td>异构微博数据抽取、解析、存储平台
运行平台：Windows 7，Apache Felix OSGi Framework</td></tr>
<tr><td>测试站点</td><td colspan="2">http://felix.torchz.net:3000/console</td></tr>
</table>

针对性能的测试，包括单一平台数据抽取的性能测试，以及多平台同步抽取的性能测试。两者各进行了 24 小时。

（1）单一平台数据抽取、解析、存储性能测试

单平台数据抽取以腾讯微博开放平台的 API 数据接入方式为例。

测试结果如图 6.24 所示。

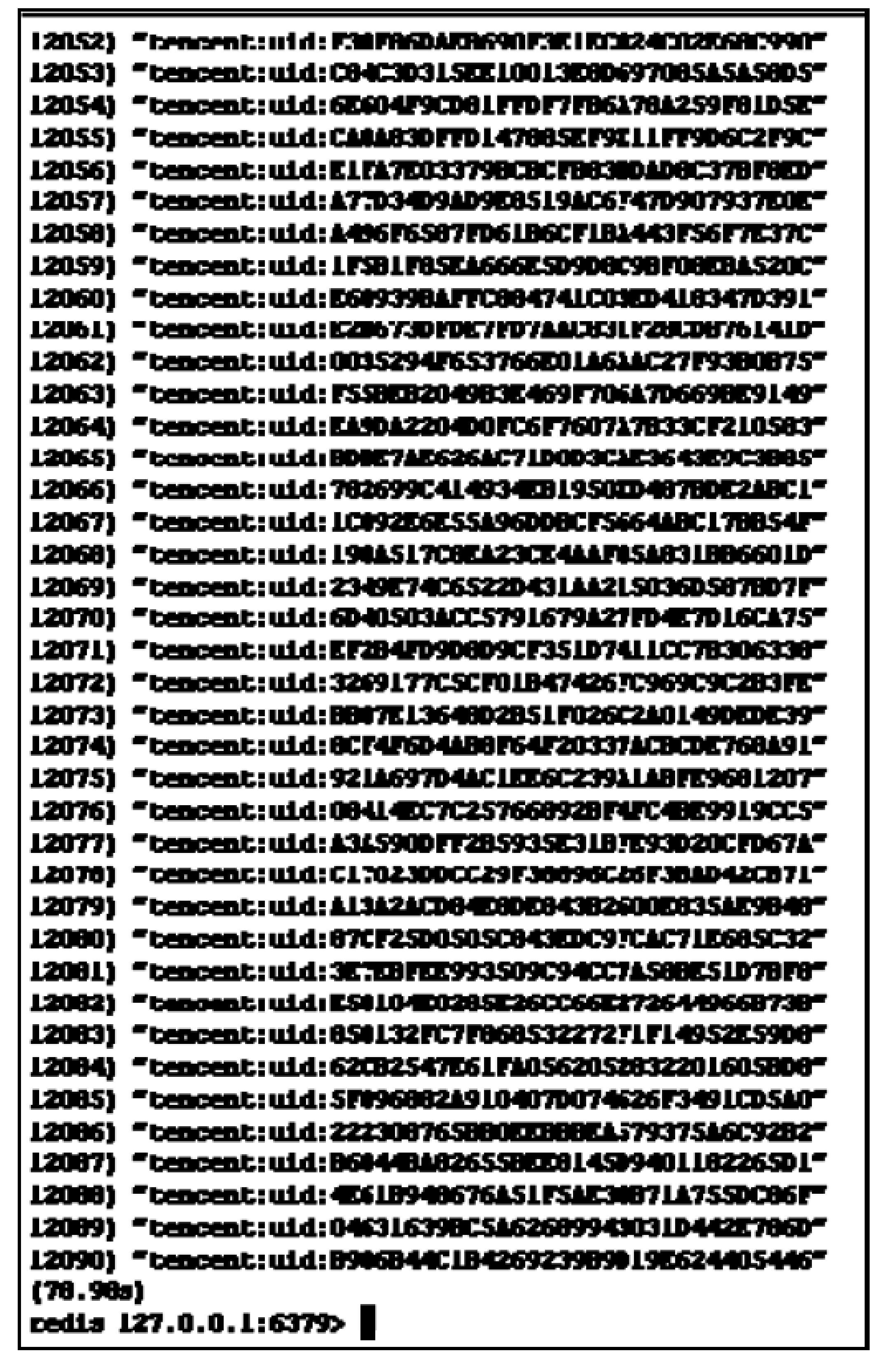


图 6.24　腾讯微博数据抽取测试结果

24 小时内抽取数据累计 12090 条，约 7 秒每条，考虑到 API 接入频率限制每小时 600 条单 IP 请求，该抽取插件良好地完成了既定目标。

（2）多平台数据抽取、解析、存储性能测试

多平台数据抽取测试包含腾讯、新浪、Twitter、和讯等多平台数据接入方式。

测试结果如图 6.25 所示。

```
59924) "tencent:uid:0841AEC7C257668928F4FC4BE9919CC5"
59925) "twitter:uid:27779611"
59926) "twitter:uid:827591610"
59927) "sina:uid:2704747357"
59928) "tencent:uid:A3A5900FF2B5935E31BFE93D20CFD67A"
59929) "tencent:uid:C17023DDCC29F38896C26F3BAD42CB71"
59930) "tencent:uid:A13A2ACD84E8DE8438260DE835AE9B48"
59931) "tencent:uid:87CF25D0505C843EDC9FCAC71E685C32"
59932) "sina:uid:2439981140"
59933) "twitter:uid:101153576"
59934) "tencent:uid:3E7EBFEE993509C94CC7A58BE51D7BF8"
59935) "twitter:uid:1347931885"
59936) "twitter:uid:584299982"
59937) "twitter:uid:38799432"
59938) "sina:uid:1068596273"
59939) "sina:uid:2422499791"
59940) "sina:uid:2149247197"
59941) "tencent:uid:E58104E0285E26CC66E272644966B738"
59942) "tencent:uid:858132FC7F868532272F1F1495 2E59D8"
59943) "twitter:uid:75179467"
59944) "tencent:uid:62CB2547E61FA056205283220160 5BD8"
59945) "sina:uid:2029169153"
59946) "sina:uid:1799285321"
59947) "sina:uid:3207485334"
59948) "sina:uid:3364749134"
59949) "twitter:uid:604943276"
59950) "sina:uid:3204926023"
59951) "sina:uid:3086693362"
59952) "tencent:uid:5F096882A910407D074626F3491CD5A0"
59953) "sina:uid:2747065111"
59954) "sina:uid:1871949182"
59955) "twitter:uid:635545417"
59956) "twitter:uid:1055418402"
59957) "tencent:uid:222308765B80EEB8BEA579375A6C92B2"
59958) "twitter:uid:96855642"
59959) "tencent:uid:B6844BA8265 5BEE814 5D9401182265D1"
59960) "tencent:uid:4E61B948676A51F5AE3887 1A755DC86F"
59961) "tencent:uid:04631639BC5A626899430 31D442E786D"
59962) "tencent:uid:B986B44C1B4269239B9D19E624405446"
(353.83s)
redis 127.0.0.1:6379>
```

图 6.25　多平台数据抽取测试结果

24 小时内抽取数据累计 59962 条，约 1.4 秒每条，考虑到网络传输瓶颈限制，本系统平台在服务器端的数据抽取已达到高效高速的需求，适当通过增加服务器台数改善网络环境可以很简单地提高抽取效率。

3. 遗留缺陷说明

系统依然仍存在的缺陷和需要考虑的 bug 如表 6.2 所示。

表 6.2　系统存在的缺陷

序号	缺陷描述	所属模块	遗留原因	影响分析
1	异常处理不成熟 存在由网络故障引起的中断	数据抽取插件模块	SSL 请求超时或服务器返回 503 错误	终止爬取逻辑，从当前节点重新开始
2	数据结构缺失造成解析结果异常	数据解析插件模块	原始数据不完整，解析系统容错性不足	自动检验并绕过不存在的数据部分

6.6 本章小结

本章对系统进行测试，并对测试结果进行分析。在系统实现的过程中，严格按照测试驱动开发，为各模块编写了单元测试用例，并以此推动开发的进行。各项功能测试用例，根据需求规约设计，从用户角度考量，对各项功能，包括数据接入、插件管理等与用户直接接触的模块做了测试。性能测试方面，分别对单一平台和多平台数据做了测试，确认系统可用性高、伸缩扩展性强且高效、低资源占用的特性。

经测试证明，本系统完成了既定目标，可以实现需求确定的各项功能，并且在抽取效率方面，性能可观。

至此我们为舆情监控系统建立了一个可伸缩的异构数据采集平台，通过这个平台可以便捷地更新插件以适应异构数据结构的变化，同时也可以对功能进行扩充，并且通过动态部署的方式，保证本系统的持续运行和高可用性。

在伸缩性方面，该平台不仅支持对系统的高层插件的更新和增加，还可以对系统自身功能进行扩展或者性能增强，这些都是依赖于满足 OSGi 规范的服务接口设计框架，通过这种模式，我们可以获得一个高效伸缩性强的数据采集途径，为舆情监控系统的后续分析处理，提供统一格式的源数据，减少后续数据梳理的难度和整个系统的复杂程度。

第 7 章

采用HBase实现海量小型XML文档的存储与检索

本章主要介绍了采用 HBase 实现海量小型 XML 文档的存储与检索，结合实际问题，对该系统的功能性需求和非功能性需求进行了详细描述，逐步深入分析，给出系统的概要设计及详细设计，并给出系统关键点的实现。通过本章的学习，读者可对采用 HBase 数据库进行大数据的存储与检索处理有一定的理论与设计经验。

7.1 应用背景

近年来，随着互联网技术的飞速发展，特别是物联网技术的兴起，网络数据的信息量正在爆炸性增长。随着信息量的飞速增长，对数据的存储和检索的要求也越来越高。无论是政府、高校、科研机构或者企事业单位的数据管理系统，还是网络论坛、SNS 网站等应用，或者是“智慧城市”、“智能物流”等物联网后端管理平台，都必须解决飞速增长的数据资源以及网络用户不断增加所导致的技术难题，传统的数据存储解决方案在面对海量数据时显得力不从心。本章的内容来源于某“物联网数据管理平台”实际项目，致力于解决海量小型 XML 文档的存储与检索问题。

网络数据资源的存在方式多种多样，但以 XML 为代表的半结构化数据，以其对逻辑和内容的合理糅合以及灵活的组织方式，在应对千变万化的应用需求中更具优势。特别是在物联网环境下，数据大多为具有一定意义的数字或文本，而 XML 数据以其不拘于结构约束并能良好表述信息内容的优势，已经成为物联网中数据传输的核心标准，目前流行的诸如 BITXML、PML 等物联网数据交换标准都是以 XML 为基础。然而，数以百万计的传感器网络或智能终端时时刻刻在产生数据，针对固定某一种应用，产生的数据往往是格式固定或大致相同，单个数据文件一般都较小，但数量是庞大的、海量的，这使得物联网后端的数据处理平台需要能够应对海量的小规模 XML 文件的存储和查询检索工作，但传统的解决方案在处理规模和性能等方面存在着瓶颈。

目前国际对 XML 数据的存储和检索的研究主要集中在三个方面：第一种是以关系数据库为核心的数据存储和检索方式，这种方式一般需要将 XML 数据通过一定的映射方式转换为关系模

型，并分表存储在关系数据库中，针对 XML 中的节点进行编码，通过关系约束将 XML 文件按照关系模型分表存储。这种方式能够有效实现数据存储和关键字检索，并支持通过 SQL 语言进行查询，但对海量数据支持不足，当文件数和数据量上升到一定程度，其性能瓶颈越发凸显，此外由于采用关系映射，一定程度上会造成信息丢失。第二种是以 Native XML 数据库为核心的数据存储和检索方式，这种方式采用类似文档式的存储模型，能够较完整保留原 XML 文件中的所有信息，并能够完美支持 XQuery 等面向 XML 的查询语言，但基于文件式的存储方式，仍然无法有效解决海量数据下的存储和检索。第三种则是在分布式环境下，采用以 NoSQL 数据库为核心的存储和检索方式，这种方式以其良好的可扩展性为立足点，能够对海量数据提供高性能的存储和查询支持，逐渐成为近年来研究的一个热点。

然而，由于一般分布式环境的复杂性，带来了诸如负载平衡、灾备恢复等问题，在部署和维护上也较为复杂。在此基础上，云计算技术开始飞速发展。2007 年底，IBM 推出蓝云（Blue Cloud）计算平台，并首次明确提出云计算技术的概念。一般情况下，云计算是指通过将计算能力和所处理的信息分布在大量的分布式计算机上，而非本地计算机或远程服务器中，在此基础上完成资源的存储和计算处理，其核心思想是应用计算模式决定底层计算模型。Google 是最早应用云计算平台的厂商之一，它提出了著名的 Bigtable 存储模型和对应的 Map/Reduce 计算模型，用于解决海量数据的存储和检索问题，取得了巨大的成功。近年来，云计算技术飞速发展，目前已经成为构建基于海量信息存储和检索的分布式应用的主流计算技术。

由于云计算技术在海量数据存储和检索中的优势，采用云计算平台作为物联网后端的数据管理平台是一种趋势，因此，如何将物联网终端所产生的海量小型 XML 数据有效地存储在云计算环境中，并能够支持高效的数据检索，具有非常明显的现实意义。

根据以上的背景分析，基于云计算平台的海量 XML 数据存储和检索技术具有非常大的发展前景，因此，有必要在云计算平台上对海量小型 XML 文档的存储与检索技术进行深入研究。

7.2 需求分析与总体设计

7.2.1 需求分析

需求分析就是研究用户具体需要得到的形式化的需求结果，需要在完全理解用户对软件需求的完整功能的基础上取得。用户需求包括功能需求和非功能需求，其中功能需求描述一个系统必须提供的功能和服务，而非功能需求描述一个用户体验很好的系统的其他特征、特点和约束条件，现对本系统进行功能及非功能需求分析。

1. 功能性需求分析

针对项目需求分析文档，待处理的数据具有以下特征：

- 数据源数量多且分布广。传感器作为物联网中的数据采集装置，数量非常大，而在空间上则呈现稀疏分散特性。因此，数据源采集到的数据需要反馈到一个统一的数据处理平台上，需要一个支持分布式计算和存储的架构。
- 单个数据文件小但总数据量大。每个传感器返回给数据处理平台的单个数据文件很小，只包含了单位时间内采集到的数据，但传感器数量可能达到百万级，并且数据更新较快。因此需要选用的数据存储系统能够支持海量数据，并同时对小文件的处理具有较高的性能。
- 根据实际情况，传感器采集的数据可能会增加。在本项目中，由于系统运行期间，可能会对传感器传回的数据结构根据实际情况进行一些调整，例如在环境监测中新参数的引入等。因此需要后端存储系统的数据模式较为灵活，主要是能够适应数据项的添加。
- 支持模式无关（Schema-free）的 XML 数据。本项目的数据来源主要有两类，除了传感器采集到的数据外，还有一类是遗产性数据，由于数据文件数目比较大，因此希望能够不需要事先提取所有数据文件的模式，而能够在处理数据文件的同时动态调整存储结构。

针对以上特点，本文所提出的海量小型 XML 数据存储和检索平台应该满足以下设计目标：

- 能够支持海量的 XML 数据的存储 并能较好地适应单个文件在 KB 级别的大规模小 XML 文件的应用场景。
- 系统应具有高可扩展性，考虑到分布式系统中数据一致性问题对数据安全的影响（特别是在军事系统中），因此存储系统最好具有强一致性。
- 平台的数据存储接口应该支持模式无关的 XML 文件的存储，能够适应新的 XML 文档中增加的节点的存储。
- 支持高性能的数据检索操作，并具有友好的数据检索接口。

根据项目背景和设计目标，经过本书前面章节中对大数据的存储和检索技术的调研和分析，项目人员决定选用 HBase 分布式存储系统作为后端存储的数据库，采用 XQuery 作为前端的用户查询接口。

选用 HBase 作为存储数据库的主要是由于：第一，HBase 作为 Hadoop 云计算环境的存储系统，以 HDFS 作为底层文件系统，具有高可扩展性；第二，HBase 本身采用强一致性模型，同时也具有不错的可用性；第三，HBase 定义的块结构为 64KB，能够有效地存储小文件；第四，HBase 采用 Key/Value 键值对存储，具有较高的性能，并且原生支持 MapReduce 编程框架。

选用 XQuery 作为前端检索查询语言，则主要是考虑到 XQuery 在 XML 查询领域的通用性和易用性。采用 XQuery 作为查询语言，一方面是基于 XQuery 在 XML 查询方面的丰富的语义及通用性，另一方面，XQuery 语言规则简单，可以使用户不需要太多额外的学习就能直接使用本文所实现的平台。

经过需求分析可知，海量小型 XML 文档存储和检索平台需要完成 XML 数据库的基本功能：

- 存储海量 XML 文档到 HBase 集群中。
- 利用 XQuery 语句进行数据的查询检索。
- 用户管理控制用户对数据表的可见度。

因此，系统组成部件应如图 7.1 所示。

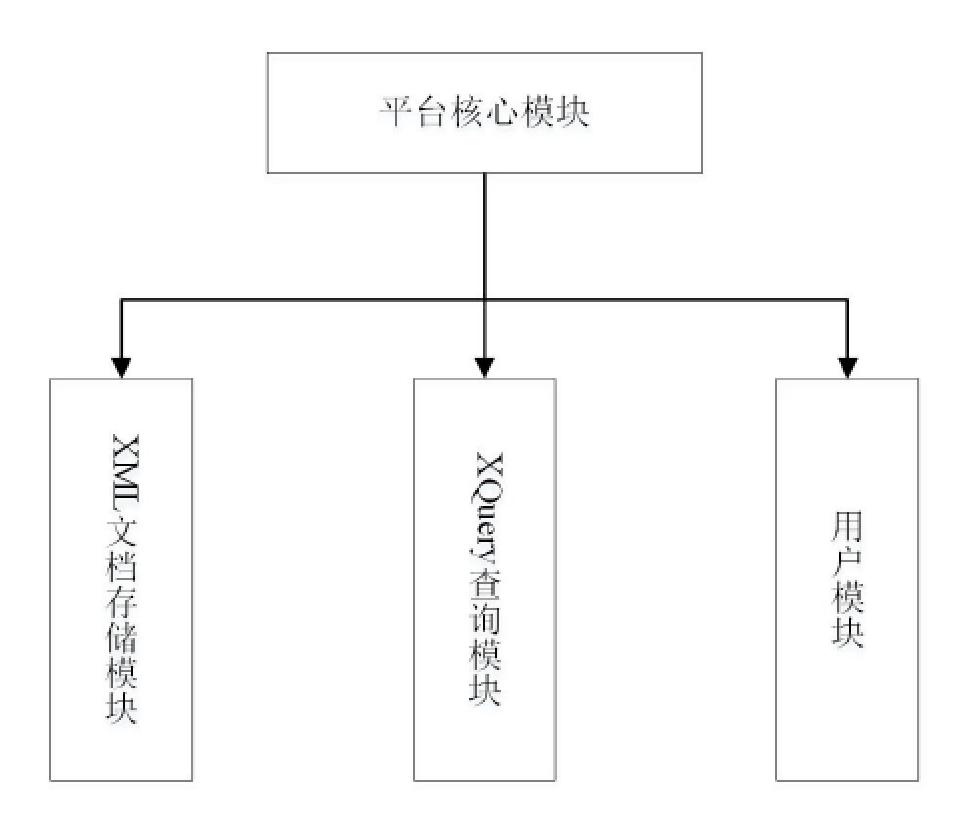


图 7.1　系统组成部件

其中，系统的核心功能是对海量XML文档的存储和利用XQuery为前端查询语言的检索功能，同时系统还要有用户登录、用户管理等功能。

用例图主要用来描述“用户、需求、系统功能单元”之间的关系。它展示了一个外部用户能够观察到的系统功能模型图。用例图帮助我们以一种可视化的方式理解系统的功能需求。本系统的用例图如图 7.2 所示。.

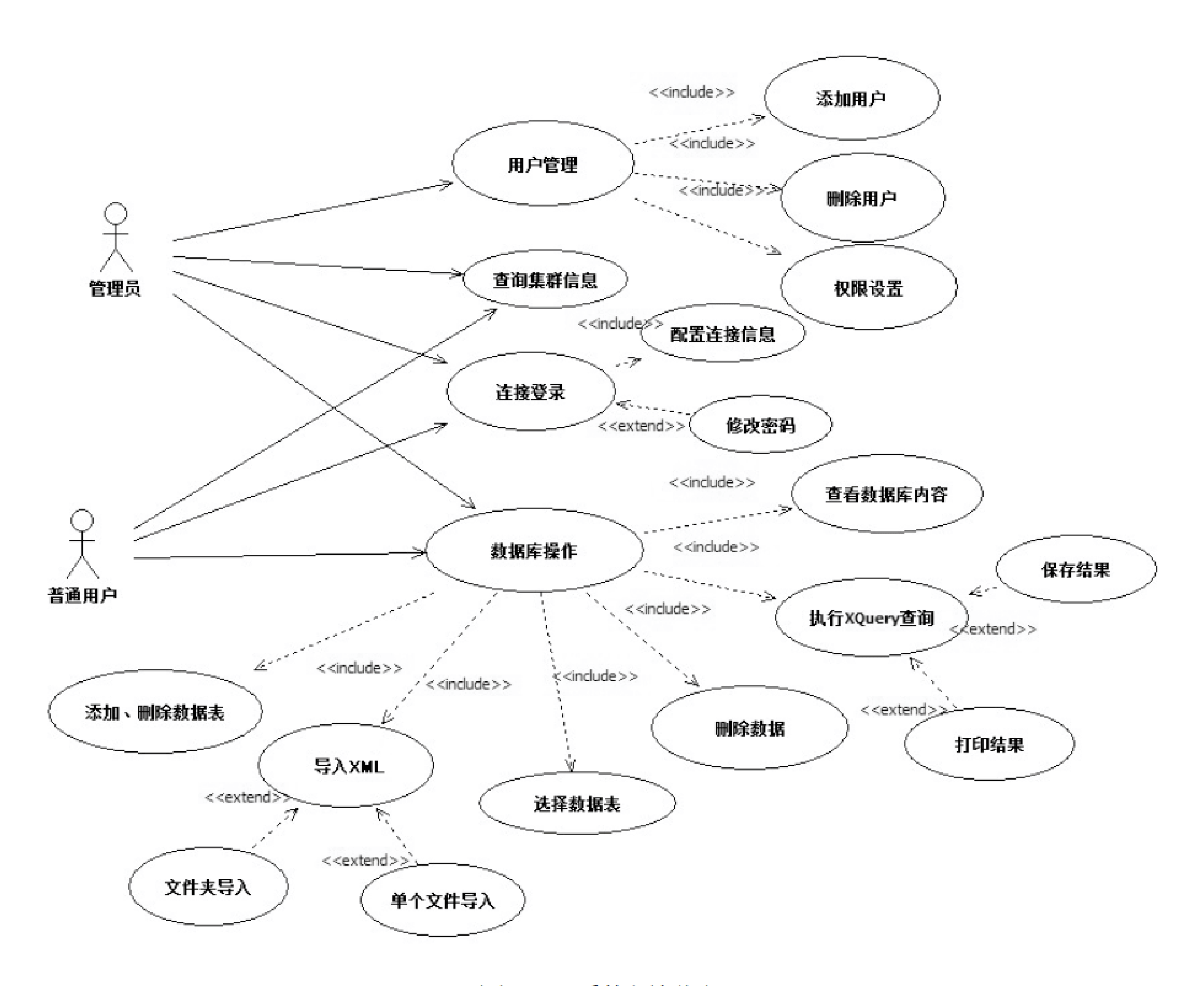


图 7.2　系统用例图

从用例图中可以看出该系统拥有两类用户：普通用户和管理员用户。管理员具有最高权限，除了能够进行有关数据的所有操作，还能够进行用户管理操作，用户管理权限包括添加用户、删除用户和针对 HBase 中的每个数据表对用户赋予不同的权限，权限分为对数据表可读、可读写和不可见三种。普通用户只能进行数据有关的操作，并且在具体操作时检查其是否具有相关操作的权限。数据有关的操作有创建删除数据表、导入数据、执行 XQuery 查询语句等。

系统上下文数据流图，是用来建立初始的系统范围的过程模型，也称为环境模型。图 7.3 是根据系统用例图绘制的本系统的上下文数据流图。从系统上下文数据流图可以看出，本系统的外部代理分别是普通用户和管理员；内部就是海量 XML 文档存储和检索平台；系统需要响应普通用户和管理员的连接登录、数据操作等事务和针对管理员的用户管理事务；系统必须根据各种操作进行响应；系统的外部存储就是 HBase 集群服务器。

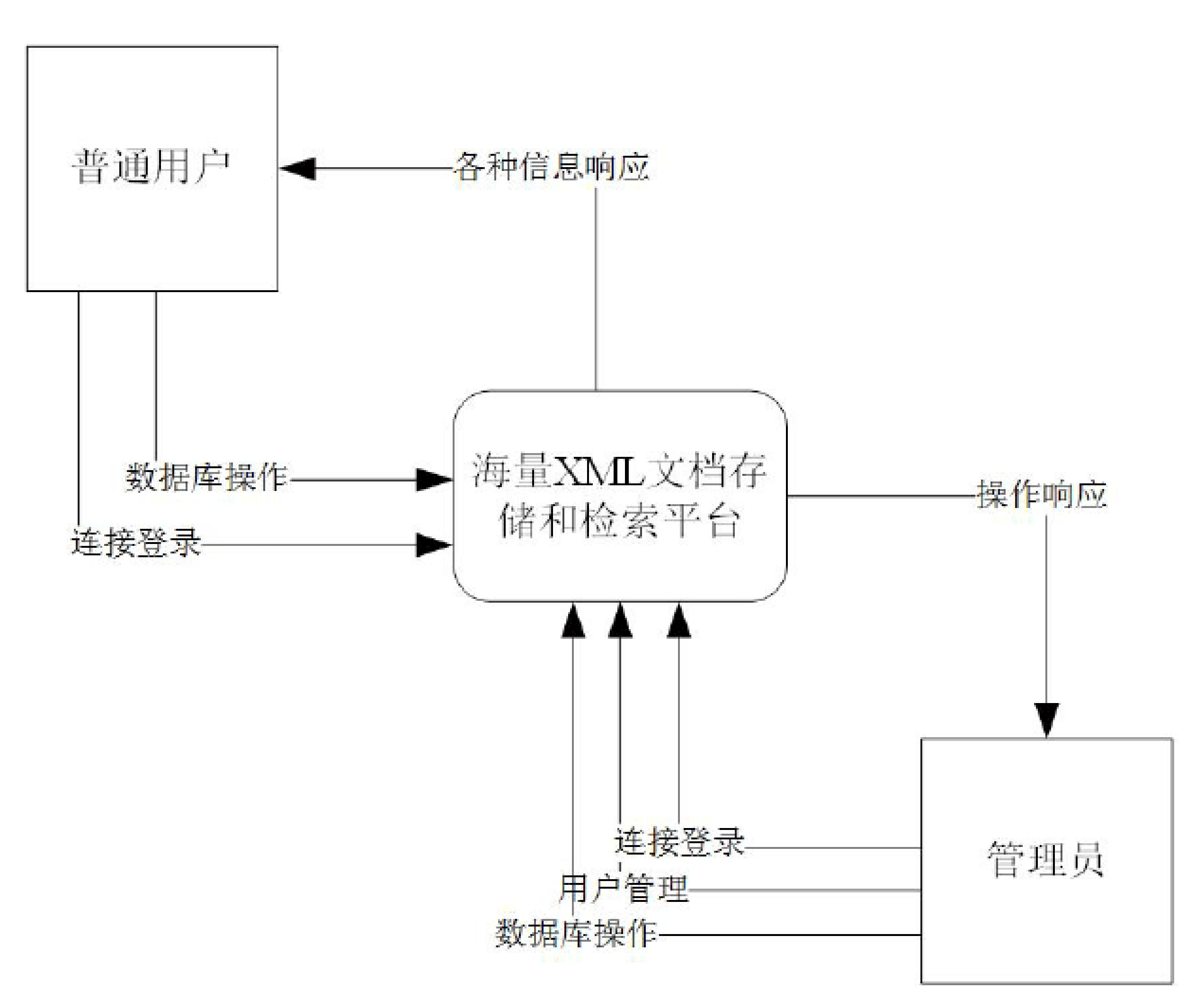


图 7.3　系统上下文数据流图

下面对本系统的重点用例采用用例表格的方式进行分析。

表 7.1 描述了用户连接登录用例。

表 7.1　连接登录用例表

用例名称	连接登录
用例标示符	uc1
优先级	高
参与者	普通用户、管理员
用例描述	用于普通用户和管理员连接登录系统
前置条件	远程 HBase 集群已启动

（续表）

基本流程	1. 编辑 HBase 服务器配置文件 server.conf（本步骤为可选操作） 2. 启动本软件系统 3. 单击连接登录按钮，弹出登录界面。在登录界面显示从配置文件读取的信息，同时用户可以手动编辑。填写用户名和密码 5. 单击“登录”按钮
其他流程	单击“取消”按钮登录界面消失，取消登录操作
异常流程	系统连接服务器错误或者用户名密码错误系统弹出消息对话框进行提示
后置条件	为普通用户和管理员生成初始界面

该用例用于普通用户和管理员的连接登录操作。用户提供远程 HBase 服务器的用户名和密码信息连接登录系统，连接登录成功进入系统后才能进行后面的针对海量 XML 文档的操作。当登录者是普通用户时界面只加载数据处理模块并根据权限列出用户可见的数据表；当登录者是管理员时，则加载数据处理界面和用户管理模块并列出系统中的所有数据表。

表 7.2 描述了存储 XML 文档用例。该用例和表 7.3 描述的 XQuery 查询用例是本系统的最核心的用例。存储 XML 文档用例描述了向 HBase 中导入海量的小型 XML 文档。

表 7.2　存储 XML 文档用例

用例名称	存储 XML 文档
用例标示符	uc2
优先级	高
参与者	普通用户、管理员
用例描述	用于将 XML 文档导入 HBase 数据库中
前置条件	用户成功登录，且用户对数据表有可写的权限
基本流程	1. 用户通过单击界面上的“上传文件”或者“上传文件夹”按钮或者在数据表上单击右键选择 2. 通过文件选择窗体进行选择 XML 文件或者文件夹 3. 对 XML 文档进行信息抽取，将 XML 文档定义为 XML 文档树，对树的节点（元素或属性）进行编码 4. 将 XML 文档与 HBase 映射，存储编码和对应的元素或属性值 5. 存储 XML 文档结束
其他流程	无
异常流程	当用户不具有对数据表可写的权限或者上传文档中出现错误时进行对话框信息的提示
后置条件	无

表 7.2 描述了存储 XML 文档用例，通过该用例用户完成对海量 XML 文档的存储，在具体存储 XML 文档操作时分为单个文件导入和整个文件夹导入，存储过程中涉及对 XML 文档内容的编码和 XML 文档与 HBase 数据库映射等操作，属于系统的核心功能之一。

表 7.3 描述了 XQuery 查询的用例，用例中以 XQuery 为前端查询语言实现对存储在 HBase 中的海量 XML 进行查询。

表 7.3　XQuery 查询用例

用例名称	XQuery 查询
用例标示符	uc3
优先级	高
参与者	普通用户、管理员
用例描述	以 XQuery 为前端查询语言对存储在 HBase 中的海量 XML 文档进行查询操作
前置条件	用户成功登录，且用户对数据表是可见的
基本流程	1. 选择将要查询的数据表 2. 在查询输入框中输入 XQuery 语句 3. 进行词法分析和语法分析，生成语法树 4. 遍历语法树，结合 HBase 的过滤器等操作，完成具体语句的查询动作 5. 在结果输出框中显示查询结果，并显示本次 XQuery 查询所用的时间
其他流程	用户完成 XQuery 语句的查询，可以单击“保存”按钮，将查询结果保存到本地文件中
异常流程	当输入的 XQuery 语句存在语法错误时，在结果输出框中显示具体的错误信息供用户参考
后置条件	无

XQuery 查询用例首先对 XQuery 语句进行解析，然后结合对 HBase 的操作完成查询操作。对于海量 XML 数据的查询是非常耗时的，为了获得良好的性能，要充分利用 HBase 自身的特点，例如过滤器、批操作等。同时在实现的过程中还要考虑由于海量数据带来的内存溢出等问题。

表 7.4 描述了用户管理用例。

表 7.4　用户管理用例

用例名称	用户管理
用例标示符	uc4
优先级	高
参与者	管理员
用例描述	管理员对该系统的用户进行管理
前置条件	当前登录者是管理员

（续表）

基本流程	1. 管理员在用户列表上单击右键并在菜单中选择“添加用户”可以实现添加用户操作 2. 在某个用户名上单击右键并在菜单中选择“删除用户”可以实现删除操作 3. 选择某个用户，在权限界面针对每个数据表进行赋权操作
其他流程	在添加用户时管理员需要设置用户的初始密码
异常流程	添加已存在用户时进行提示
后置条件	用户管理操作后将信息更新服务器记录

只有管理员用户才有权限进行用户管理，可以添加用户、删除用户和针对每个数据表进行赋权操作。用户的管理信息保存在 HBase 服务器端。

2. 非功能性需求分析

非功能性需求是指软件产品为满足用户业务需求而必须具有的除功能需求以外的特性。软件产品的非功能性需求通常包括系统的可靠性、可维护性、可扩充性以及对技术和业务的适应性等。下面总结了针对本系统的非功能需求。

- 易用性：本系统要提供良好的交互界面，符合大部分用户的使用习惯，能够在短暂时间内学会该系统的使用方法。
- 可靠性：要求系统能够持续可靠地正常运行，并具有一定的容错性，当用户进行一些非法操作或者错误操作时能够进行信息提示并中断操作。
- 性能：在执行 XQuery 语句进行海量 XML 数据检索时，要比同等条件下传统数据库和原生 XML 数据库用时更短。

7.2.2　总体设计

好的架构设计能够提高软件系统的整体质量，有利于软件系统的维护、升级和修改，对整个系统有着深远的影响，可以让开发人员减轻解决复杂问题的负担和精力。

1. 系统架构图

本系统以分布式数据库 HBase 为存储介质，通过将 XML 文档编码映射存储在 HBase 集群服务器上完成对海量小型 XML 文档的存储，使用 XQuery 为前端查询语言，通过解析并结合 HBase 的操作完成对海量 XML 文档的检索；系统还要提供良好的用户界面。通过上述描述发现该系统的层次非常分明，适合使用三层模型作为整体架构，图 7.4 是该系统的系统架构。

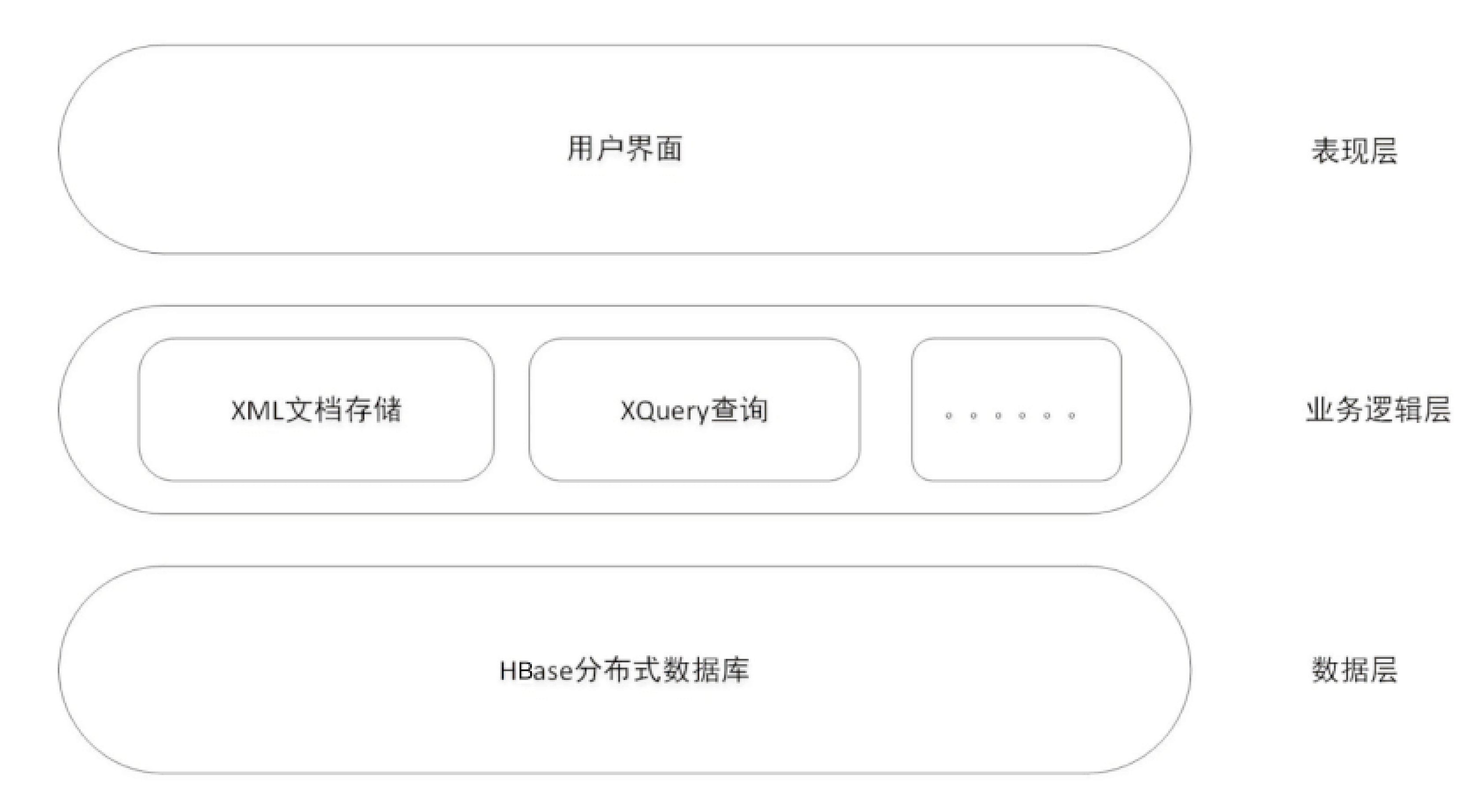


图 7.4　平台架构图

由图 7.4 可以看出，本系统的三层模型分别为表现层、业务逻辑层和数据层。具体来说，前端用户界面以及存储和查询接口为表现层，用于显示数据和接收用户输入的数据，为用户提供一种交互式操作。中间 XML 文档的存储、XQuery 查询等系统核心处理部分构成业务逻辑层，该层是系统的核心，主要处理用户的请求，负责对数据层的操作。底层 HBase 分布式数据库是系统的数据层，用于存储具体的 XML 文档数据。

该系统使用三层模型的优点有：

- 降低层与层之间的依赖，开发时可以只关注整个结构中的其中某一层。
- 使系统的结构更清楚，分工更明确，有利于后期的维护、升级和功能扩展。
- 有利于系统各层逻辑的复用，对于本系统，复用业务逻辑层可以为以后其他项目提供应用接口。
- 有利于标准化。

系统使用三层模型有以上优点的同时也有些缺点，例如由于采用了多层处理的方式，使得系统的多个业务不能直接访问 HBase 数据库，而通过业务逻辑层访问从而造成了系统性能的降低。但是总体来说采用三层模型开发该系统是利远远大于弊的，该架构是非常合适的。

2. 系统总体设计

由需求分析可知，海量小型 XML 数据存储和检索平台主要可以分为三个模块：数据存储模块、XQuery 查询模块和用户模块。各模块主要功能如下。

（1）数据存储模块

数据存储模块负责存储海量 XML 数据到系统中，其输入数据是大量的小型 XML 文档，而结果则是将 XML 文档中的数据按照一定的结构顺序存储在 HBase 数据库中。数据存储模块会根据输入的 XML 文档，按照 XML2HBase 模型映射机制，对当前 XML 文档生成一个映射表，这是一个双向映射表，能从一个 XML 节点的路径映射到 HBase 的列，同时也能够从 HBase 的列找到其 XML 节点路径。

（2）XQuery 查询模块

具体来说，XQuery 查询模块又包括两部分：查询解析和数据检索。查询解析模块主要负责将用户输入的 XQuery 查询语言经过解析后，按照 XQuery 查询语言的语义，构造相应的语法树，遍历 XQuery 语法树，根据相应的语义完成对数据库的检索，并按照一定的规则对结果进行排序并返回给用户。XQuery 查询模块的核心问题是如何在保证检索准确的情况下高效快速地得到检索结果，由于采用 Hadoop 作为分布式环境，因此可以采用 MapReduce 并行计算模型，在一定程度上加速检索效率。

（3）用户模块

用户模块主要是对系统进行多用户支持，提高系统的利用率，同时预防某些用户的恶意操作。与用户模块有关的操作有用户登录、管理员管理用户、管理用户权限等操作。系统为了完成以上的功能或操作需要在服务器端保存用户的用户名、密码、针对每个 XML 数据表的权限等信息，当用户登录或者对用户管理时对 HBase 上的数据进行查找或修改。

根据图 7.4 的系统总体架构可以看出，数据存储和 XQuery 查询作为系统两大核心模块，其设计如图 7.5 所示。

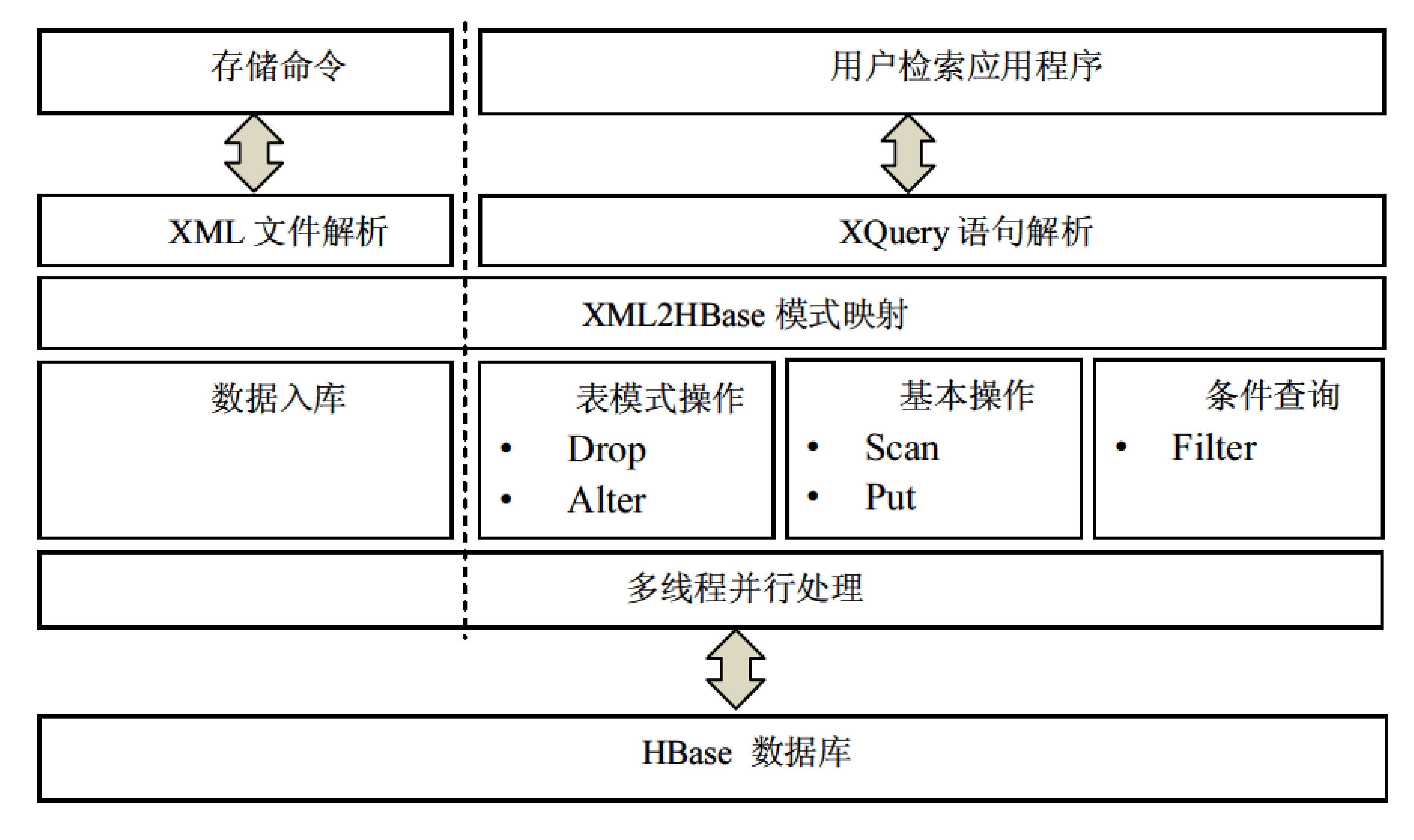


图 7.5　平台核心模块

由图 7.5 可以看出，从存储方面来看，用户调用前端 XML 存储 API，中间层对作为输入源的 XML 文档进行解析，然后根据 XML2HBase 模式映射，将 XML 路径映射到 HBase 特定的列，并存入数据库。从检索方面来看，用户在前端编写 XQuery 查询语句，中间层首先解析 XQuery 语句，构建相应的语法树，并将相关数据根据 XML2HBase 的映射替换为 HBase 中的列，再调用特定的操作函数来检索数据库。在部分操作上引入 MapReduce 并行编程框架会极大地提升存储和查询速度。

除了各自的解析模块外，都使用到 XML2HBase 模式映射层，这是因为用户提交的 XML 文档和 XQuery 查询语句中，都是基于原生的 XML 文件的操作，用户看到的是 XML 的数据模型。

但本文中的海量 XML 数据存储和检索平台事实上是一个分布式的系统，后端采用 HBase 作为数据库，因此 XML 数据也会最终转化为 HBase 所支持的数据模型。因此，在 XML 数据模型和 HBase 数据模型之间，需要有一种映射转化模型，支持双向的数据模型映射，而这里的 XML2HBase 就是本文提出的一种数据模型映射机制。

7.3 相关技术介绍

通过前面的分析，与本项目的解决方案相关的技术有 XML 相关技术、XQuery 相关技术、HBase 相关技术以及 JavaCC 相关技术。在本书前面章节已经对 HBase 有过相关介绍，所以本节不再赘述。

7.3.1 XML 相关技术

1. XML 简介

在 W3C 发表的 XML 规范中详细定义和解释了 XML 相关标准。XML 通常被当作一种特殊的半结构化数据，它具有半结构化数据不规则结构、隐含模式信息、可以描述数据的结构信息等特点，同时也有自己的特征。

下面为一个 DBLP 文件的片段，这是一个典型的 XML 文件。XML 文件由元素构成，元素由开始标记和结束标记来标识，元素中可以使用属性来区别。其中 dblp 和 article 都是元素，而 article 元素中的 key 和 mdate 则是属性。<dblp>是开始标记，</dblp>是结束标记，开始标记和结束标记是成对出现的，并且，标记可以嵌套，但其嵌套遵循就近匹配的原则，相同的元素起始标记和结束标记对称出现。例如<markA><markB>text</markB></markA>，markB 是 markA 的子元素，因此 markB 的开始标记和结束标记都被 markA 包含在内。

```
<?xml version="1.0"?>
<dblp>
<article key="journals/ijcc/Sasikala11" mdate="2012-05-31">
<author>P. Sasikala</author>
<title>Cloud computing: present status and future implications.</title>
<year>2011</year>
<journal>IJCC</journal>
</article>
<article key="journals/pvldb/Matsudaira10" mdate="2010-09-23">
<author>Paul Matsudaira</author>
<title>High-End  Biological  Imaging  Generates  Very  Large  3D+  and  Dynamic
```

```
Datasets.</title>
    <year>2010</year>
    <journal>PVLDB</journal>
    </article>
    </dblp>
```

可以从上述 XML 文档中发现，一个 XML 文档由一个最外层的元素及其子元素组成，XML 元素之间存在着嵌套关系，两个元素之间存在上下层关系，而同层元素之间具有一定的顺序。元素的属性为元素提供一些额外的内容信息，元素的属性在元素的开始标记中给出，一般表示为“属性名=‘属性值’”，其中属性值必须使用单引号或者双引号括起来。一个元素的属性数目不限，但属性名不能相同。

一般而言，从数据模型角度来看，可以将 XML 数据建模为树结构。XML 文档中的元素是树结构中的节点，树中的边表示元素之间的父子关系。一个元素的属性一般建模为该元素节点的一个孩子节点。树形结构是本文对 XML 建模的基础，在 XML 解析、处理过程中，这种数据模型非常有效。

2. XML 解析

XML 解析就是解析 XML 文档的过程，将 XML 文档转换成一种计算机可以利用的数据结构，方便对 XML 文档的后续操作。目前，基本的解析方式有两种，一种是 SAX，另一种是 DOM。SAX 是基于事件流的解析，DOM 是基于 XML 文档树结构的解析。

DOM（Document Object Model）也称为文件对象模型，是 W3C 组织推荐的处理可标记语言的标准编程接口。DOM 可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。DOM 处理 XML 文档时将 XML 文档存入内存，将页面上数据和结构进行树形表示，用户可以使用具体的 DOM 操作 API 访问树的节点，并可以对数据进行修改、添加和删除等操作。

SAX（Simple API for XML）也称为 XML 简单编程接口，是一个通用的事件驱动的 XML 解析器，它是一个轻量型的解析器，已得到广泛的应用，SAX 在解析 XML 文档时，不在内存中创建完整的 XML 文档树，而是基于事件驱动，因为 SAX 事件驱动的本质，处理文件通常会比 DOM 风格的解析器快，占用内存少。

7.3.2　XQuery 语句

1. XQuery 简介

XQuery 为 XML Query，是 W3C 所制定的一套标准，用来从类 XML 文档中提取信息，类 XML 文档可以理解成一切符合 XML 数据模型和接口的实体。XQuery 相对于 XML 的关系，等同于 SQL 相对于数据库表的关系。XQuery 被设计用来查询 XML 数据。XQuery 被构建在 XPath 表达式之上，XPath 即为 XML 路径语言（XML Path Language），它是一种用来确定 XML 文档中某部分位置的语言，主要实现 XQuery 语言的路径表达式，是 XQuery 的重要组成部分。

XQuery 使用路径表达式在 XML 文档中通过元素进行导航。下面的路径表达式用于在 books.XML 文件中选取所有的 title 元素：doc("books.XML")/bookstore/book/title。XQuery 使用谓语来限定从 XML 文档所提取的数据，下面的谓语用于选取 bookstore 元素下的所有 book 元素，并且所选取的 book 元素下的 price 元素的值必须小于 30：doc("books.XML")/bookstore/book[price<30]。

XQuery 的规则是：

- XQuery 对大小写敏感。
- XQuery 的元素、属性以及变量必须是合法的 XML 名称。
- XQuery 字符串值可使用单引号或双引号。
- XQuery 变量由“$”并跟随一个名称来进行定义，如$bookstore。
- XQuery 注释被（: 和 :）分割，例如，（: XQuery 注释 :）

2. FLWOR 表达式

许多（不是全部）XQuery 查询结构是 FLWOR 表达式，FLWOR 语句相当于 SQL 中的 select-from-where 结构，FLWOR 由 for、let、where、order by 和 return 这几个表达式中使用的关键字的首字母组成。FLWOR 允许对结果进行操作、变换、排序。其中：

- for 语句用于指明查询中使用的文档、声明变量、并把变量绑定到某个节点序列上。
- let 语句用于对 for 语句指定范围的所有变量进行赋值，若没有 for 语句，则对当前的所有变量进行赋值。
- where 语句用于指定条件，只有那些满足条件的变量才能被 return 语句返回。多个条件可以通过 and 和 or 连接。
- order by 语句用于把 for，let 语句指定的满足 where 条件的记录进行排序。
- return 语句指定返回的结果及结果的形式。

7.3.3 XML 检索技术

1. XML 检索技术概述

针对 XML 的树形结构，目前研究者已经提出了多种基于路径表达式的查询语言，其中最著名的就是 XQuery 和 XPath。为了能够有效地支持 XML 的内容和结构的查询，很多研究专家都提出了针对 XML 数据的一些编码模式和索引技术。

一种编码方案需要从两方面来衡量其有效性：节点之间关系的判断和查询满足特定结构关系的节点集。现有著名的编码模式如 Dewey 编码、Dietz 编码、Li-Moon 编码、OrdPath 编码等。其中 Dietz 编码无法对 XML 树结构添加节点，除非对树的所有节点重新编码，Li-Moon 编码则为节点插入预留了空间，但占用的资源较多。

2. XML 编码技术

（1）Dewey 编码

Dewey 编码最早使用在一般的知识分类系统中，是一种前缀编码。Dewey 编码方法是，在处理 XML 文档时，对每一个节点都分配一个编码，这个编码是从文档的根节点到当前节点的路径。这个路径每一个数值都是一个祖先节点的编码，因此其上层节点的编码是当前节点编码的前缀，而上层节点与孩子节点之间用分隔符“.”隔开。

如图 7.6 是一个 Dewey 编码的实例，这棵树 T 中有一个节点 η，则 η 的 Dewey 编码是 C(η)。若 η 节点具有一个孩子节点 μ，则 μ 节点的 Dewey 编码 C(μ)= C(η).n，其中的 n 是节点 μ 在节点 η 的所有孩子中的序号。

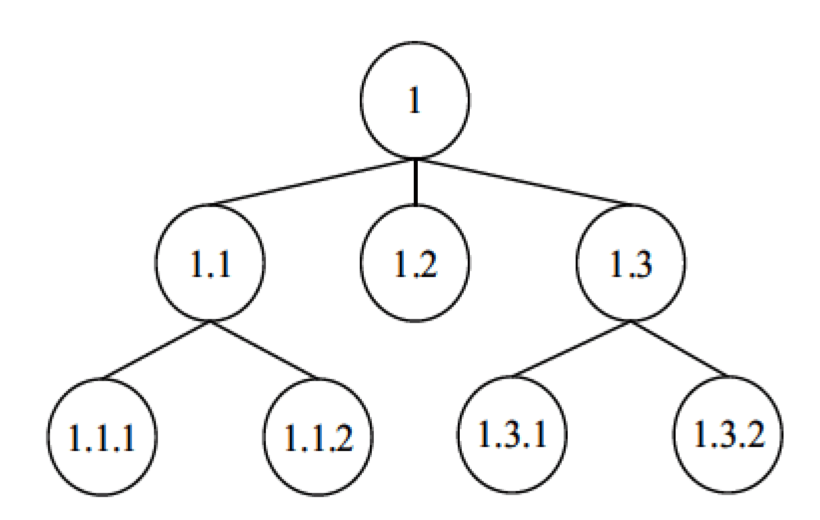


图 7.6　一个 Dewey 编码的实例

Dewey 编码的前缀编码特性使得判断任意两个节点间的结构关系变得很容易，相同前缀必定具有相同的祖先节点。然而文档更新时，新加入的节点会影响到部分已经编码的节点。

（2）OrdPath 编码

OrdPath 是一种类似于 Dewey 编码的编码方式，也是前缀编码的一种。OrdPath 中对于父子关系的节点，编码规则也是将父节点的 OrdPath 编码进行扩展得到子节点编码。举例来说，如果 1.5.3 是父节点的编码，那么 1.5.3.9 必定是该节点的孩子节点。OrdPath 编码与 Dewey 编码的主要区别在于，OrdPath 中将偶数节点保留，未来需要插入新节点时使用偶数节点作为父节点。

例如图 7.7 所示的树结构。初始编码方式，树 T 中有一个节点 η，则 η 的 OrdPath 编码是 C(η)。若 η 节点具有一个孩子节点 μ，则 μ 节点的 OrdPath 编码 C(μ)= C(η).(2n+1)，其中的 n 是节点 μ 在节点 η 的所有孩子中的序号。如果需要在初始编码结构中插入一个新的节点，例如节点在 1.3 和 1.4 之间，那么首先在 1.3 和 1.5 之间的位置插入一个偶数编码节点 1.4，然后将新节点作为 1.4 节点的孩子插入树结构中，编码为 1.4.1。

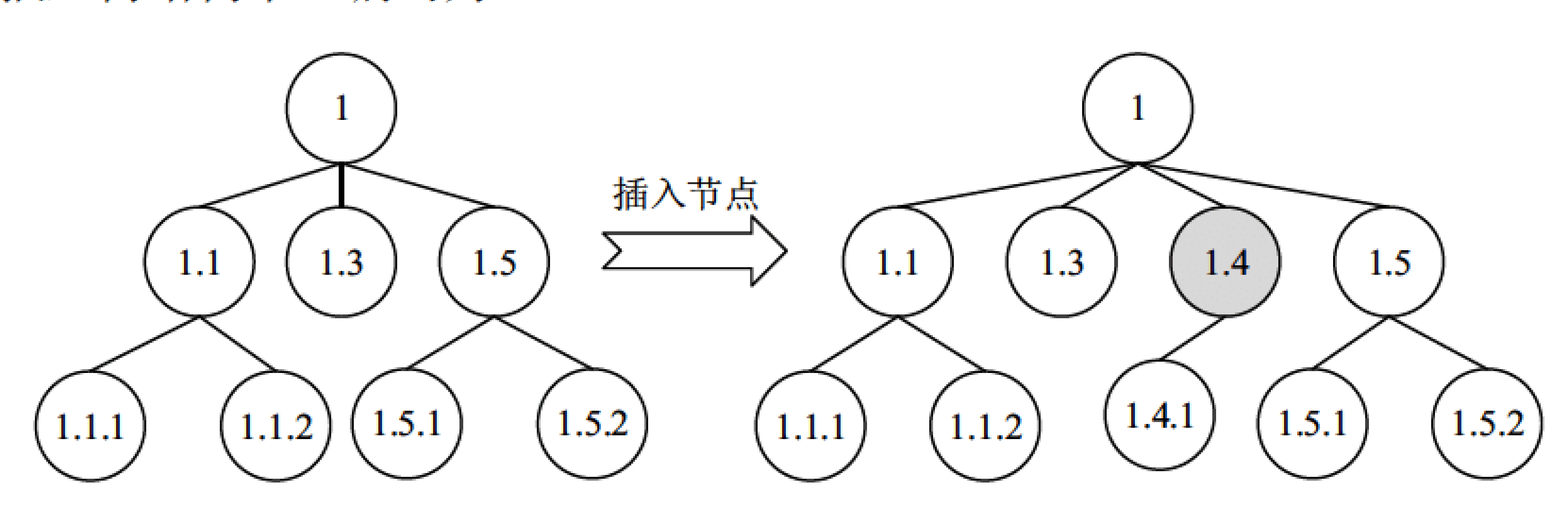


图 7.7　OrdPath 节点编码示例

ORDPath 编码在保留前缀编码优异的节点关系结构判断能力的基础上，能够在插入新节点时无需改动已编码节点。但其缺点是对同路径重名节点支持不够，并且容易导致 XML 结构树的深度随着插入节点而变的非常大。

7.3.4 云计算和 HBase

1. 云计算与 Hadoop

随着互联网技术的飞速发展，特别是面向分布式和大数据的新模式的出现，云计算作为一种新兴的技术越来越被科研和企业界所重视。2010 年 9 月，卡耐基梅隆大学软件工程研究所大系统（System of Systems）研究小组的 Grace Lewis 以白皮书的形式给出了云计算的定义：云计算是一种采用虚拟化、面向服务的计算和网格计算等已有技术的大规模分布式计算范型，为获取和管理大规模 IT 资源提供一种不同的方式。

目前，包括 Google、IBM、微软和 Amazon 等公司都在大力发展云计算技术，分别在基础平台即服务（Infrastructure as a Service，IaaS），平台即服务（Platform as a Service，Paas）和软件即服务（Software as a Service，SaaS）等不同服务模式上提出了优秀的产品。

Hadoop 是 Apache 开源组织的一个分布式计算框架，可以在大量廉价的硬件设备组成的集群上运行应用程序，为应用程序提供了一组稳定可靠的接口，旨在构建一个具有高可靠性和良好扩展性的分布式系统。随着云计算技术的逐渐流行与普及，该项目被越来越多的个人和企业所运用。Hadoop 项目的核心是 HDFS、MapReduce 和 HBase，它们分别是 Google 云计算核心技术 GFS、MapReduce 和 Bigtable 的开源实现。其系统环境图如图 7.8 所示。

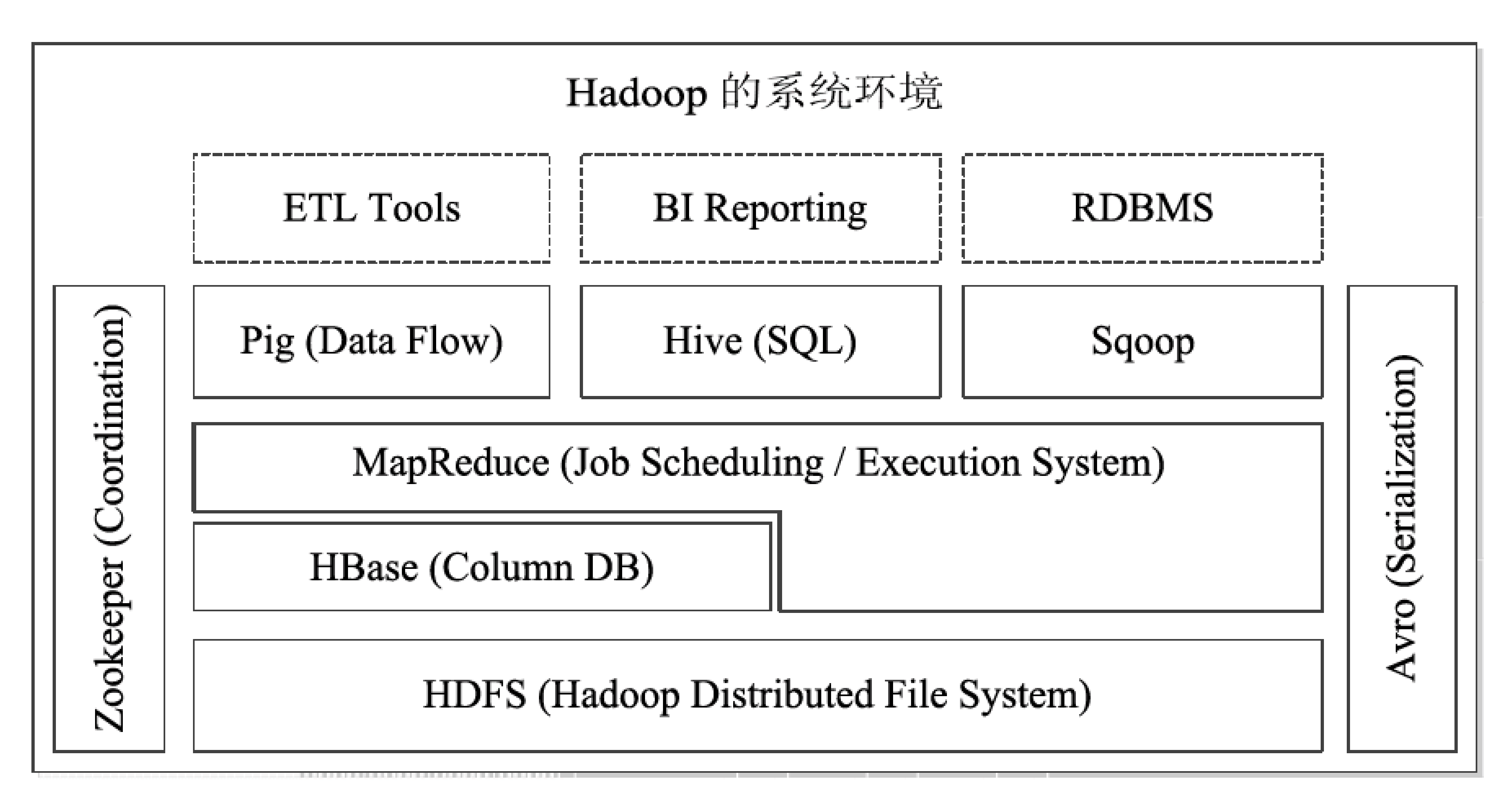


图 7.8 Hadoop 的系统环境图

Hadoop 的核心组件有以下 3 个。

（1）HDFS（Hadoop 分布式文件系统）

HDFS 在整个 Hadoop 体系结构中处于最基础的地位，HDFS 分为三个部分：客户端、主控节

点（Namenode）和数据节点（Datanode）。Namenode 是分布式文件系统的管理者，主要负责文件系统的命名空间、集群的配置信息和数据块的复制信息等，并将文件系统的元数据存储在内存中；Datanode 是文件实际存储的位置，它将数据块（Block）信息存储在本地文件系统中，并且通过周期性的心跳报文将所有数据块信息发送给 Namenode。HDFS 采用 master/slave 架构。一个 HDFS 集群由一个 Namenode 和一定数目的 Datanodes 组成。Namenode 是一个中心服务器，负责管理文件系统的名称空间（namespace）以及客户端对文件的访问。集群中的 Datanode 一般是一个节点一个，负责管理它所在节点上的存储。

（2）MapReduce

为了解决处理海量原始数据的复杂问题，Google 设计了一个新的抽象模型 MapReduce，使用这个抽象模型，程序员只要表述他想要执行的简单运算即可，而不必关心并行计算、容错、数据分布、负载均衡等复杂的细节。

与传统的分布式程序设计相比，MapReduce 封装了并行处理、容错处理、本地化计算、负载均衡等细节，还提供了简单而强大的接口。通过这个接口，可以把大数据量的计算自动地并发和分布执行，使之变得非常容易。另外，MapReduce 也具有较好的通用性，大量不同的问题都可以简单地通过 MapReduce 来解决。海量数据的运算大多数都包含这样的操作：在输入数据的“逻辑”记录上应用 Map 操作得出一个中间 key/value 集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，从而达到合并中间的数据，得到一个想要的结果的目的。使用 MapReduce 模型，再结合用户实现的 Map 和 Reduce 函数，就可以非常容易地现大规模并行化计算；MapReduce 模型自带的“再次执行”（re-execution）功能，也提供了初级的容灾方案。

MapReduce 的运行模型如图 7.9 所示。

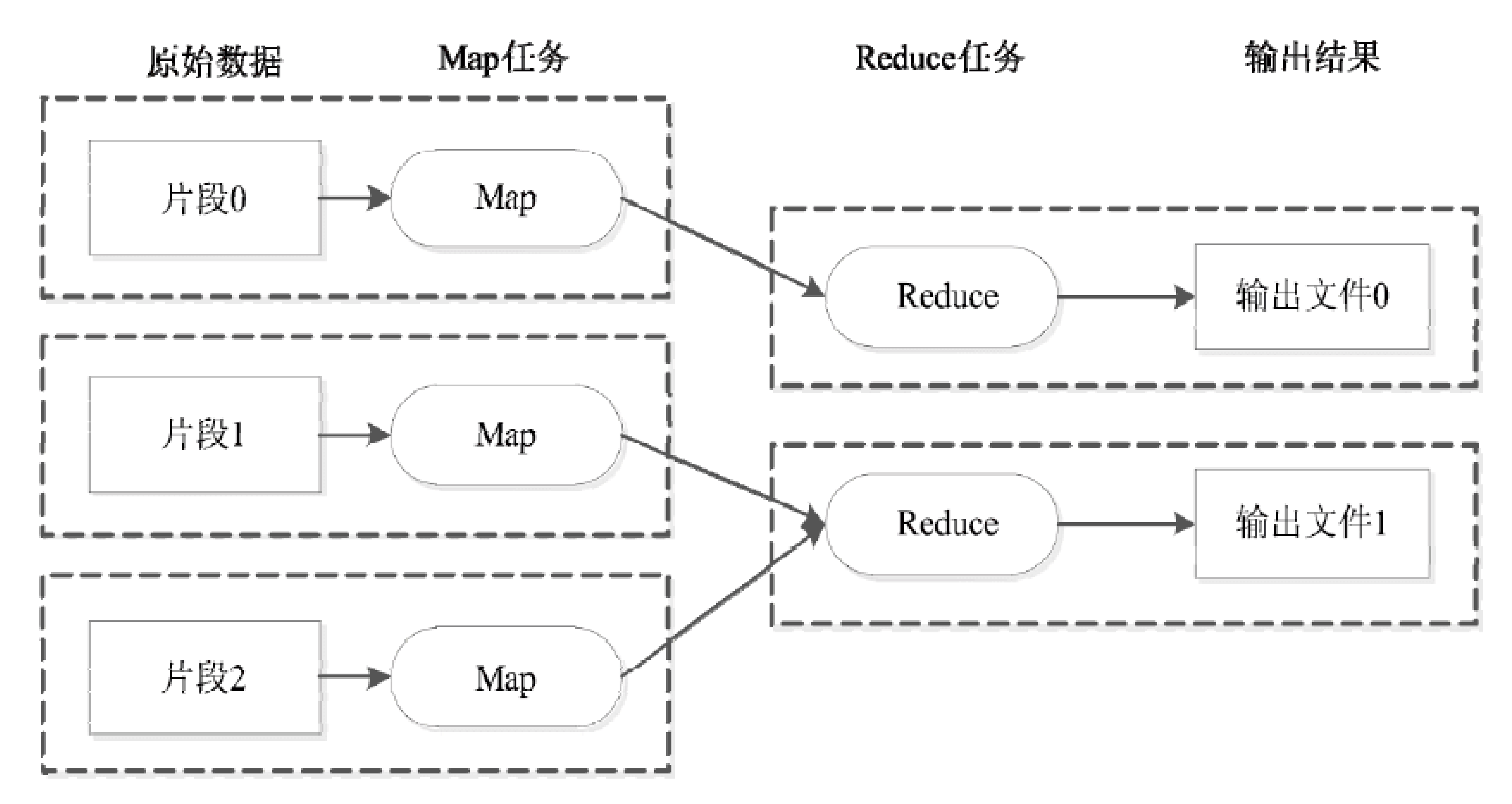


图 7.9　MapReduce 的运行模型

（3）HBase

HBase 即 Hadoop Database，是一个构建在 HDFS 上，面向列的开源分布式数据库系统，是 Google Bigtable 的开源实现。HBase 适用于对超大规模数据集的实时随机读写。它的设计初衷就是在廉价的商用硬件集群上处理一些超大数据表，这些表的规模通常达到数十亿行和数百万列。

HBase 并不是关系数据库，它并不支持 SQL，HBase 提供了一组简单的 API 接口，用于存取和管理数据。

2. HBase 数据库

HBase 是一个 NoSQL 数据库。NoSQL 有时也被认为是 Not Only SQL 的简写，是对不同于传统的关系型数据库的数据库管理系统的统称。NoSQL 是非关系型数据存储的广义定义。它打破了长久以来关系型数据库与 ACID 理论大一统的局面。NoSQL 数据存储不需要固定的表结构，通常也不存在连接操作。在大数据存取上具备关系型数据库无法比拟的性能优势。

按照 CAP 理论，HBase 的设计目标是高可扩展性和强一致性，HBase 确保分布在网络不同节点上数据的一致性，因此降低了对可用性的要求。CAP 理论最早是由 UC Berkeley 大学的 Eric Brewer 提出的。CAP 理论指出，一个分布式系统最多只能满足以下三种需求中的两种。

- 一致性（Consistency）：指分布式系统中一个数据在多个节点的备份都是相同的，即某一数据在集群中不同节点中内容一致。
- 可用性（Availability）：每一个操作无论是请求失败或成功，总是能够在确定的时间内得到响应。
- 分区容忍性（Partition Tolerance）：在出现网络分区（例如断网）等情况时，系统中任意信息的丢失不会影响系统的继续运行（除非整个网络都出现故障）。

这个定理使得数据库架构师在设计数据库系统时，无需再去费尽心机地尝试使系统同时满足一致性、可用性和分区容忍性这三种需求，而可以集中精力按照系统的需求，设计合适的系统架构来满足这三者中的两个。

HBase 部署在 Hadoop HDFS 上，使用 Master-Slave 架构。Master 节点负责与 Client 端交互，完成数据 Region 的分配，并将用户的数据访问请求转到具体的存储节点上。RegionServer 作为从属节点，负责存储具体的数据，一个 RegionServer 会管理多块 Region 数据。HBase 的集群管理通过 Zookeeper 实现。

3. HBase 过滤器介绍

关系数据库的 SQL 语句提供了丰富的查询条件，方便用户对表中的数据进行过滤。HBase 作为分布式数据库，也提供了对数据进行过滤的功能。HBase 的 Scan 和 Get 操作都能够获取指定行关键字的数据，甚至可以获取特定行、特定列和特定时间范围的数据，HBase 完成以上功能是通过过滤器完成的。HBase 提供了一个过滤器结构类 Filter，并预先实现了多个常用的过滤器。用户需要对数据进行过滤时，可以选择适当的过滤器对数据进行过滤，用户也可以通过实现 FilterBase 类构建自定义过滤器，这使得 HBase 的过滤功能具有很高的灵活性。

7.3.5 JavaCC 工具介绍

JavaCC（Java Compiler Compiler）是一个用 Java 开发的最受欢迎的语法分析生成器。这个分

析生成器工具可以读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的 Java 程序。JavaCC 可以在 Java 虚拟机（JVM）V1.2 或更高的版本上使用，它是 100%的纯 Java 代码，可以在多种平台上运行，与 Sun 当时推出 Java 的口号“Write Once Run Anywhere”相一致。JavaCC 还提供 JJTree 工具来帮助我们建立语法树，JJDoc 工具为源文件生成 BNF 范式（巴科斯-诺尔范式）文档（Html）。

下面是 JavaCC 的一些具体特点：

- JavaCC 产生自顶向下的语法分析器，而 YACC 等工具则产生的是自底向上的语法分析器。采用自顶向下的分析方法允许更通用的语法（但是包含左递归的语法除外）。自顶向下的语法分析器还有其他的一些优点，比如易于调试、可以分析语法中的任何非终结符、可以在语法分析的过程中在语法分析树中上下传值等。
- 词法规范（如正则表达式、字符串等）和语法规范（BNF 范式）书写在同一个文件里。这使得语法易读和易维护。
- 默认情况下，JavaCC 产生的是 LL(1)的语法分析器，然而有许多语法不是 LL(1)的。JavaCC 提供了根据语法和语义向前看的能力来解决在一些局部的移进-归约的二义性。例如，一个 LL(k)的语法分析器只在这些有移进-归约冲突的地方保持 LL(k)，而在其他地方为了更好的效率而保持 LL(1)。移进-归约和归约-归约冲突不是自顶向下语法分析器的问题。
- JavaCC 提供了多种不同的选项供用户自定义 JavaCC 的行为和它所产生的语法分析器的行为。
- JavaCC 提供的 JJTree 工具，是一个强有力的语法树构造的预处理程序。
- JavaCC 允许拓展的 BNF 范式——例如(A)*、(A)+等。拓展的 BNF 范式在某种程度上解决了左递归。事实上，拓展的 BNF 范式写成 A ::= y(x)* 或 A ::= Ax|y 更容易阅读。

7.4 详细设计与实现

前面已经进行了平台的需求分析描述，并给出了系统总体架构，现在开始介绍平台的具体设计和实现。海量 XML 文档存储和检索平台的核心功能是以 HBase 数据库为存储介质对海量 XML 文档进行存储，以 XQuery 为前端查询语言对存储在 HBase 中的 XML 数据进行检索，对系统的用户进行管理和权限控制。图 7.10 为抽象出的核心模块设计图。

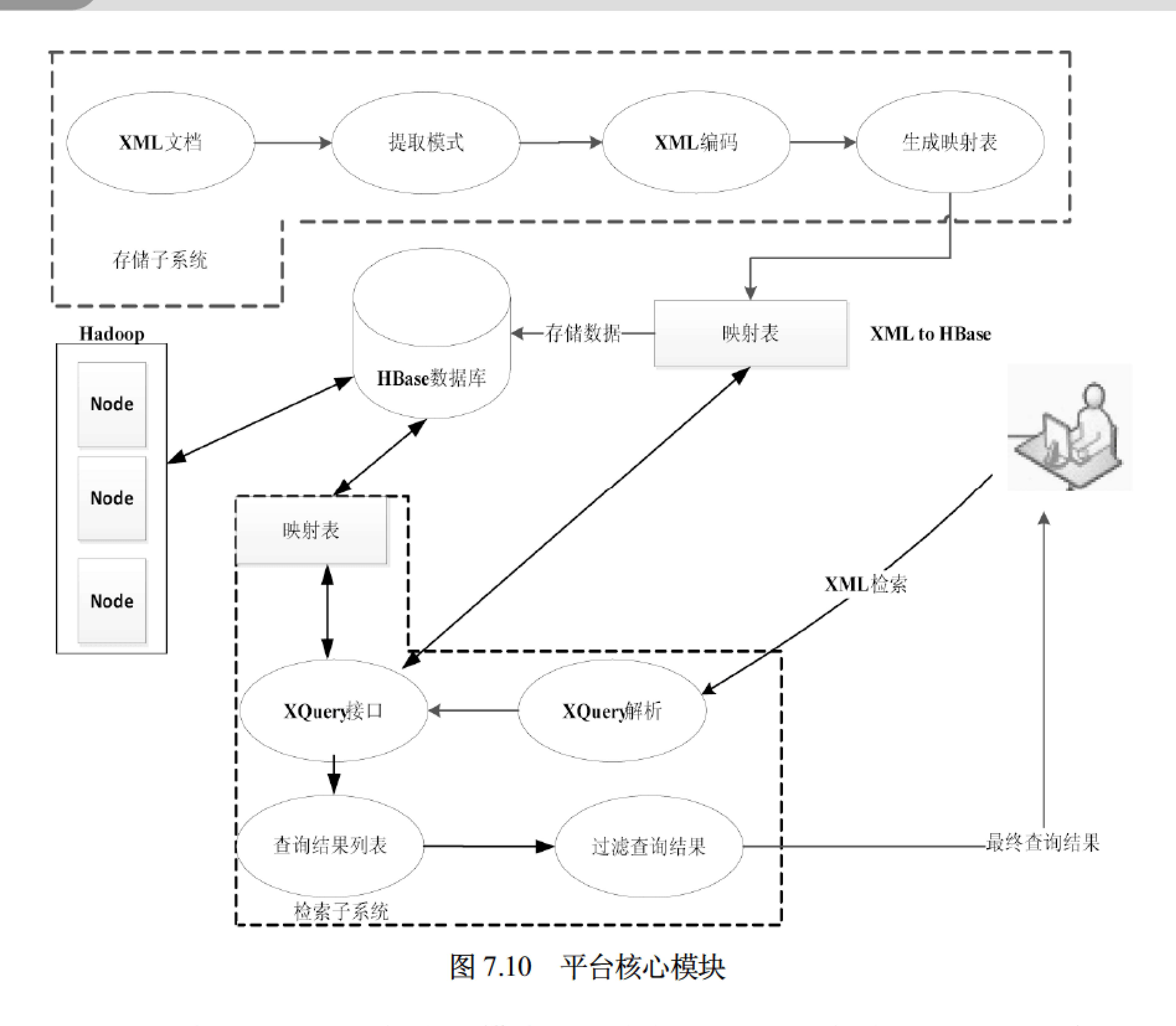


图 7.10　平台核心模块

从图 7.10 可以看出，XML 文档存储模块用于将海量 XML 文档存储到 HBase 中，XQuery 查询模块用于解析 XQuery 语句并查询 HBase。另外还有用户模块处理用户的登录、用户管理等操作。下面介绍系统的具体实现。

7.4.1　数据存储模块的详细设计与实现

由于 XML 是树结构数据模型，对树结构的查询事实上有非常成熟并且高效的算法，但由于 HBase 数据模型不同于树结构模型，一些典型并且知名的算法直接在 HBase 上的检索效率并不高，因此需要寻求一种新的映射技术。本小节提出了 XML2HBase 数据映射模型，为 XML 节点路径和 HBase 中的键建立一个双向的映射，形成如图 7.10 中的映射表，并给出了映射所采用的四路节点编码方案的编码规则。

1. XML 数据模型

W3C 的 XML 规范中，关于 XML 的定义较为复杂。事实上，相对于本项目的需求，由于传感器网络传给后端处理的数据一般都是半结构化数据，因此从 XML 模型上可以做一定程度的简化。本文建立的 XML 数据模型更强调图论上的树结构特性，这更符合通过编程接口处理得到的 XML 数据。这是因为 XML 的标准编程模型是 DOM 模型，我们的树形模型类似于 DOM 模型：把节点当作对象，把边当作对象引用。

首先需要声明两个集合：一个标签的有限字符集 Σ，一个值数据的无限字符集 P。由于标签

名需要符合 XML 规范约束，并且标签数量是有限的，例如具有 Schema 的 XML 文档等，因此Σ是一个有限字符集。而值集则包含各种字符自由组合、数字等，因此是一个无限字符集 P。

本文这样定义一颗 XML 树：

定义 7.1　一颗 XML 树 $t=\{N,E,<,\lambda,\nu\}$ 由以下部分组成：一颗有向树{N,E}，其中 N 是这颗 XML 树的所有节点的集合，$E\subseteq N\times N$ 表示这棵树所有的边；一个 N 上的偏序关系<，这种偏序关系是按照深度优先遍历的顺序得到的，因此对于任意节点 x 是节点 y 的祖先，必然满足关系 $x<y$；一个节点标记函数 $\lambda:N\rightarrow\Sigma$；一个值函数 $\nu:N\rightarrow P$。

一颗 XML 树可以表示为 t={N,E}，因为 $<,\lambda,\nu$ 等信息可以通过上下文环境得到。$\Gamma_{\Sigma,P}$ 用来表示节点标签来自集合Σ和数据值来自集合 P 的 XML 树的集合；相应的 Γ_{Σ} 表示只有节点标签属于集合Σ的 XML 树集合，例如没有定义值函数 ν 的 XML 树集合。

事实上，任何一个符合 W3C 标准的 XML 文档都可以抽象成一颗 XML 树 t={N,E}。但从数学上看，定义中给出的 XML 树的描述仍然不够精确，因为一颗抽象 XML 树能够转换为不同形态的 XML 文档。下面给出 XML 数据模型的一些更准确的定义。

（1）集合 N 可以分为四个不相交的集合，即 $N=\{d\}\cup Elmt\cup Attr\cup Txt$，其中：

- d 是 XML 树 t 的根节点，叫做文档节点。
- $Elmt$ 是所有元素节点的集合。
- $Attr$ 是所有属性节点的集合。
- Txt 是所有文本纯节点的集合。

（2）$Attr\cup Txt$ 只包含所有的叶子节点，因此，如果存在 $(n,n')\in E$，那么 $n\in\{d\}\cup Elmt$。

（3）同一个节点的属性子节点出现在其他类型节点的前面，即如果存在 $(n,n_1)\in E,(n,n_2)\in E,n_1\in Attr,n_2\in Elmt\cup Txt$，那么 $n_1<n_2$。

（4）节点标记函数 $\lambda:N\rightarrow\Sigma$ 只在元素和属性节点 $Elmt\cup Attr$ 上定义，在其他节点上没有定义节点标记函数。

（5）数据值函数 $\nu:N\rightarrow P$ 只定义在属性和纯文本节点 $Attr\cup Txt$ 上，其他节点上没有数据值函数定义。

（6）文档节点 d 的所有子节点都是元素节点 $Elmt$，即如果 $(d,n)\in E$，那么 $n\in Elmt$。

2. HBase 数据模型

HBase 中的数据存储在表（Table）中，表则由行（Row）和列（Column）构成。表中的任何一个列都归属于一个特定的列族（Column Family）。行和列的交叉点，构成表的单元格（Cell），单元格是有版本号的（versioned），默认情况下，版本号是 HBase 自动分配的，即插入单元格时的时间戳（Time Stamp）。单元格中的数据是没有类型的，直接以字节数组形式存储。

HBase 的数据被建模成一个四维的映射，表中的单元格数据可以由以下索引唯一定位：

```
(表名: string, 行关键字: string, 列关键字: string, 时间戳: int64) → 数据值: string
```

其中，表名（Table Name）是一个字符串，唯一标识一张表；行关键字（Row Key）一般指定为一个字符串，唯一标识该行，表中的行通过行关键字按照字典序排列；列关键字（Column Key）的构成格式为“列族名:标签”，列族可以理解为对列的分类，在该分类下通过标签确定一个具体的列，例如列关键字 school:class 可以表示学院列族中的班级列，每个列族都可以划分任意数量的标签。

表 7.5 显示了 HBase 数据存储的逻辑模型，事实上这是一个稀疏的表。一个表的列族作为表模式的一部分，需要在定义表时预先给出。一旦设定了表的模式和列族，在后期使用中可以随时在列族中增加新的成员，即列族的数量是无需预先定义的，在使用中可以动态扩展。例如表 7.5 中已经定义了列族 Info，当客户端在更新时提供了新的列 Info:content，那么 HBase 会在 Info 中增加新的列来存储它的值。

在物理上，HBase 表都是按照列来存储的。因此，空白单元格并不会存储在列中。HBase 会自动将表水平地划分为区域（Region），每个 Region 都是由表中行的子集构成的。Region 是 HBase 集群分布数据的最小单位，当一个表太大时会分割成多个 Region 分布在服务器集群上，集群中每个节点负责管理表的一部分 Region，因此 HBase 集群中的从属节点叫做 RegionServer。

表 7.5　HBase 中的表

Row Key	Time Stamp	Column Family	
		URL	Info
row1	t5	url: = “www.xidian.edu.cn”	Info:title = “Xidian Univ.”
	t4		Info:content = “<html…”
	t3		
row2	t2	url: = “www.xde6.net”	
	t1		Info:title = “Xidian news”

在 HBase 中，文件的存储是通过 HFile 类来实现的，HFile 也是 HBase 的文件格式。HFile 文件中存储的是有序的 Key/Value 键值对，其中所有 Key 和 Value 都是字节数组格式的。HFile 文件的结构组织是基于 Block 的，文件中存储着一到多块数据块（Data Block），此外还有零到多块元数据块（Meta Block）、一个 FileInfo 数据块、一个数据块索引（Data Block Index）块、一个元数据块索引（Meta Block Index）块以及一个尾指针（Trailer）块。

如图 7.11 为 HBase 的存储模型，其中键值对部分为 HBase 在实际存储中的数据模型，正如前面给出的四维映射。因此，可形式化地定义 HBase 的数据模型：

定义 7.2 对于一个 HBase 表 T，其内容为五元组 $T=\{K,V,\kappa,\gamma,\upsilon\}$。其中 K 表示所有键集合，V 表示所有值集合，$\kappa$ 表示键名函数 $K \to \Sigma'$，γ 表示值函数 $V \to P'$，υ 表示键到值的函数 $K \to V$。

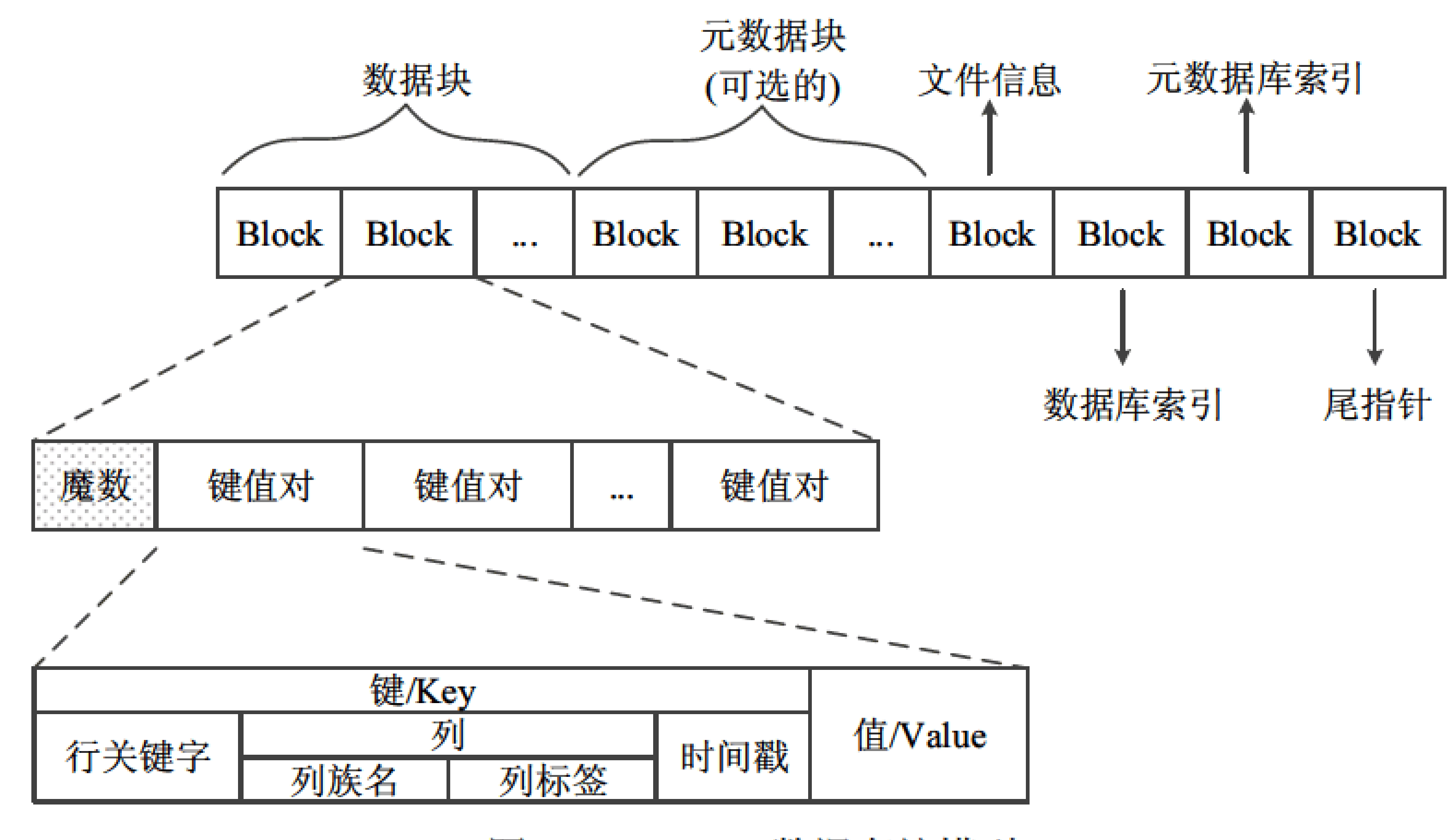


图 7.11　HBase 数据存储模型

事实上，一个 HBase 表的键集合是一个有限字符集，这里命名为 Σ'，而值集合是一个无限集合，命名为 P'。由于键集合由 HBase 规定的行、列和时间戳组成，在 HBase 中行名、列名和时间戳的命名都是有限集合，因此 Σ' 是一个有限字符集。而值集则包含各种字符自由组合、数字等，因此是一个无限字符集。

定义 7.3　对于一个 HBase 表 T，其键集合中的键 $k \in K$ 都是由一个四元组构成：$k=\{r,c,q,t\}$。其中 r 表示行关键字，c 表示列族名，q 表示列标签名，t 表示时间戳。

由定义 7.2 可以推出，对于 HBase 的表 T，其一个任意键 $k \in K$，通过键值映射函数 υ 可以找到一个值 $v \in V$。即：

$$v=\upsilon(k) \qquad \text{式（7-1）}$$

3. XML2HBase 模型映射机制

在建立了 XML 和 HBase 的数据模型后，尝试给出 XML2HBase 的映射规则。首先应该明确，数据映射是存在于 XML 文档和 HBase 数据库之间的一种转换规则，通过 XML2HBase 模型映射机制，能够得到一个双向的映射表，这个映射表能通过 XML 文档的节点路径得到该节点路径对应的 HBase 的列名，也能通过列名查找到该节点在 XML 文档中的路径。

事实上，通过 XML 文档路径进行映射的技术叫做节点编码技术，在第二章中已经介绍过了，由于本项目的设计目标中，对于输入数据的定义是模式无关的 XML 文档，因此需要对现有的节点编码技术进行一些改进。

XML 文档的文档节点 d 作为 XML 文档树结构的根节点，是唯一判定一个 XML 文档的标识，因此，这里将具有相同文档节点的 XML 文档定义为同一类文档。即这类 XML 具有相似的结构，描述着类似的内容，例如在物联网中描述地理环境参数的文档，都以<geography>标签作为根节点，而这一类 XML 文档将放在 HBase 中以 geography 为名的表中。此外，由于要处理模式无关的 XML 文档，因此 HBase 需要具有灵活的模式，方便插入操作，因此一个 HBase 表具有唯一的列族，这个列族也以文档节点 d 命名。

定义 7.4　对于文档节点为 d 的 XML 文档和 HBase 中的表和列族，具有以下映射：

$$\kappa(T_d)=\lambda(d) \qquad 式（7-2）$$

$$\kappa(c)=\lambda(d) \qquad 式（7-3）$$

式（7-2）和式（7-3）中的κ是 HBase 表中的键的名称的映射函数。λ则是 XML 节点的名称映射函数。根据 XML2HBase 映射机制，XML 文档到 HBase 的映射主要需要完成的映射如定义 7.5 所示。

定义 7.5 e是 XML 文档中的一个元素节点的路径表达式，那么e和 HBase 中的键具有如下映射关系：

$$h(e)=K_e,\quad e\in E,K_e\subset K \qquad 式（7-4）$$

$$h'(k)=e,\quad e\in E,k\in K \qquad 式（7-5）$$

式（7-4）中的h是 XML 文件中的节点路径到 HBase 中一组关键字集合的映射函数，由于 XML 的路径表达式得到的节点并不唯一，例如路径表达式/bookstore/book 的 book 标签可能有多个，因此 HBase 中也对应着多个列。值得注意的是，路径表达式事实上是和 HBase 中的列对应的，因此关键字集合K_e并不用特别指明行关键字 r，尽管对于单个 XML 文档来说 r 是确定的。此处给出 XML 文件到 HBase 表中键的行关键字的映射关系。

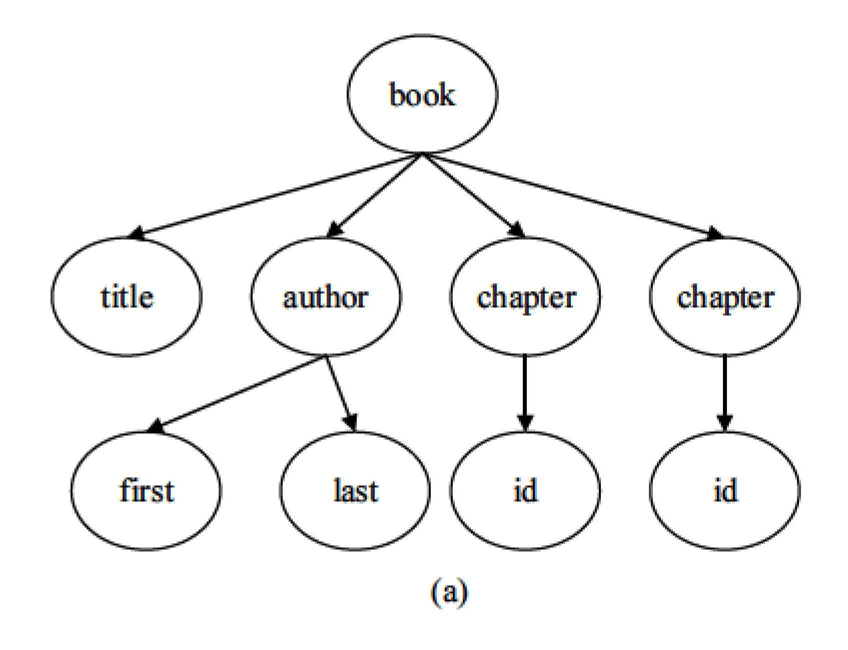
式（7-5）则是 HBase 中表的一个键k到 XML 的元素路径的映射函数h'，由于 HBase 的一个键只包含一个列，因此对应唯一一个 XML 元素。

4. 四路节点编码方案

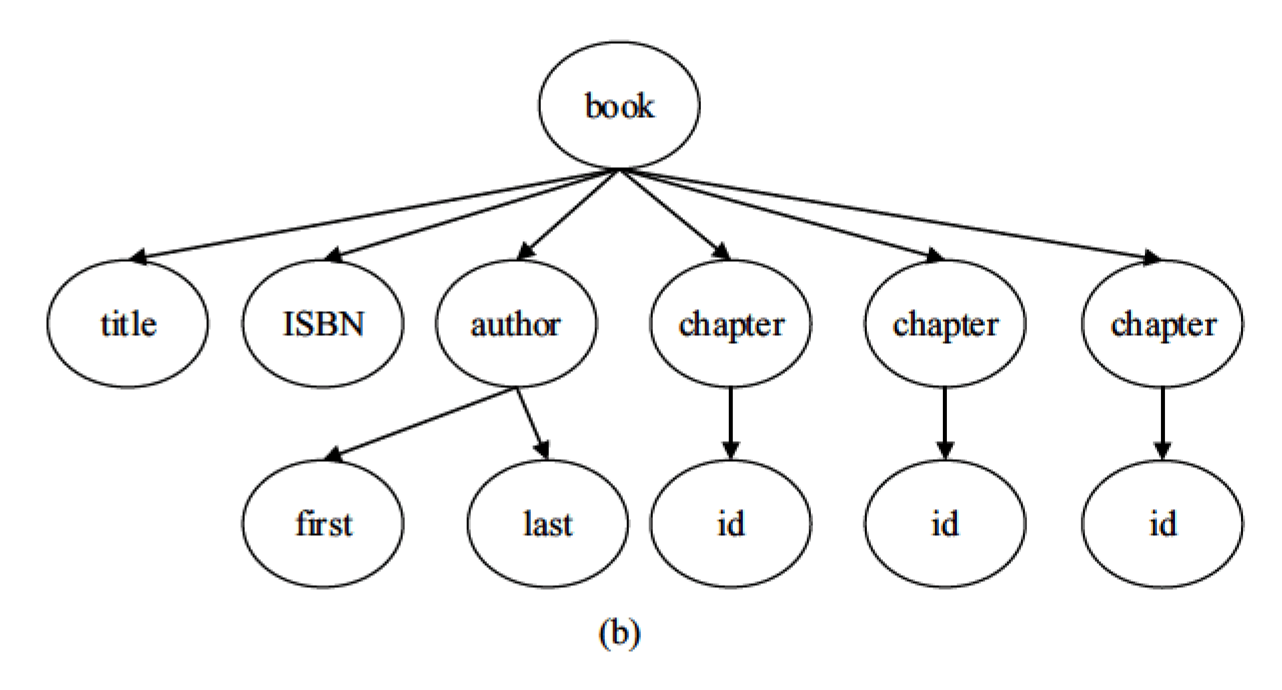
为了完成 XML 节点到 HBase 列的双向映射，这里提出一种基于路径类索引技术的节点编码方案。相比 OrdPath 编码方案，四路节点编码方案具有更高的灵活性，能够适应元素动态插入和同名节点的扩展。

模式无关的 XML 存储方式主要难点在于应对新增节点的问题，新增节点主要包括两类：一类是原本没有出现过的路径节点，如图 7.12(b)中的 ISBN 节点，在图 7.12(a)中没有出现过，需要插入到 title 和 author 节点中间；第二类则是新增的同名节点，由于同名节点可以出现任意多次，节点出现次数不固定，且没有上限，因此需要使用同名节点的动态扩充。

OrdPath 编码方式考虑了第一类新增节点情况，但并没有考虑第二类新增节点情况，而四路节点编码方式则在 OrdPath 的基础上进行扩展，并支持第二类新增节点的处理。



(a)



(b)

图 7.12　XML 结构示例

四路节点编码方式可以通过以下规则描述：

- XML 文档的文档节点作为根节点，编码为 1。
- 每个节点具有一个当前层节点编号 S_t。具有相同父节点的节点，其当前层节点编号不同，当前层编号是一个整数. 具有相同父节点的所有节点按照前序遍历顺序的访问次序 p 来命名，且：

$$S_t = 4p + 1 \quad \text{式（7-6）}$$

除根节点以外的所有节点 I，其节点编码采用其父节点的编码作为前缀，并加上 I 节点的当前层编号，中间以分隔符“.”隔开，因此节点 I 的编码

$$S_I = S_p.S_t \quad \text{式（7-7）}$$

- 元素的属性节点当作元素的子节点处理，并且属性节点位于所有其他子节点之前，属性节点的命名规则是在原名称前加“@”符号。
- 如果遍历到一个路径已经出现过的节点 I，如图 7.12(a)中最右边的 chapter 节点，已存在相同路径节点 A，则判断当前层编号比 A 节点小 1 的节点是否存在，如果不存在，则新建一个虚拟节点 B，令其当前层编号为，并将 I 节点放在 B 节点下一层；如果已存在 B 节点，则按照前面的规则，将 I 节点放在 B 节点下一层的最右边。

$$S_{tI} = \begin{cases} (S_{tA} - 1).1, & S_{tA} - 1 = \emptyset \\ (S_{tA} - 1).last(), & S_{tA} - 1 \neq \emptyset \end{cases} \quad \text{式（7-8）}$$

- 当处理一个新 XML 文档时，先序遍历这个 XML 文档中的节点，如果发现新的没出现过的节点 I，且这个节点出现在已编码的两个节点 A 和 B 之间，且 A 节点的当前层编号为，B 节点的当前层编号为，则 I 节点的当前层编号为。

$$S_{tI} = \begin{cases} (S_{tA} + 1).1, & S_{tB} = S_{tA} + 4 \\ (S_{tA} + 1).last(), & S_{tB} = S_{tA} + 1 \end{cases} \quad \text{式（7-9）}$$

即在 A 节点和 B 节点之间创建一个虚拟节点 V，并将 I 节点作为 V 节点的第一个子节点，如果 V 节点已存在，则将 I 放在 V 节点的最后。

根据上述规则，图 7.12(b)在使用四路节点编码方案后如图 7.13 所示。

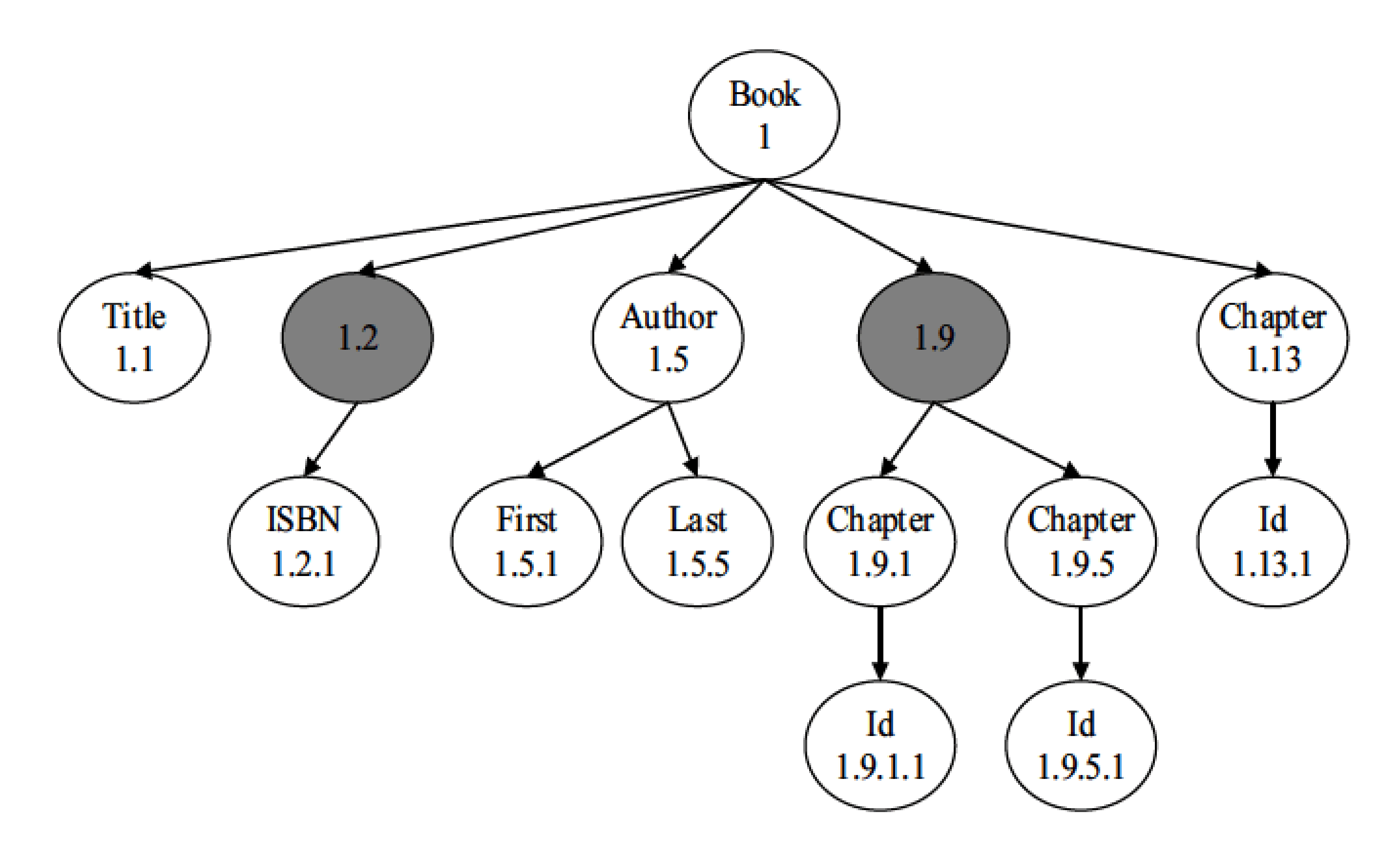


图 7.13　四路节点编码方案示例

节点 1.2 和 1.9 都是虚拟节点，虚拟节点本身并不代表任何路径，而该节点的所有孩子节点在逻辑上都是和虚拟节点在同一层的，在处理检索的时候应该将虚拟节点的所有孩子节点向上提到虚拟节点所在层来对待。

四路节点编码方案最后会产生一个映射表，是 XML 路径到 HBase 列的双向映射，根据图 7.13 会产生如表 7.6 所示的映射表。

表 7.6　图 7.13 产生的映射表

XML 路径	HBase 列名
/Book	Book:1
/Book/Title	Book:1.1
/Book/ISBN	Book:1.2.1
/Book/Author	Book:1.5
/Book/Author/First	Book:1.5.1
/Book/Author/Last	Book:1.5.1
/Book/Chapter	Book:1.9.1 Book:1.9.5 Book:1.13
/Book/Chapter/@Id	Book:1.9.1 Book:1.9.5 Book:1.13

该映射表可以采用 HashMap 的形式保存在内存中，构成节点路径索引，当查询 XML 路径时可以迅速得到对应的 HBase 列，反之亦然。

根据上述规则，可以得到一个规律，即任何编码的最结尾 S_t 是 4n+1 的节点都是原始元素节点，而以 4n+2 结尾的都是新添加的节点，以 4n 结尾的都是和 4n+1 节点相同路径的节点。此处 4n+3 结尾的节点编号并没有使用，因为目前的编码规则已经可以满足项目需求，因此 4n+3 结尾的节点编号留下来作为以后可能出现的扩展情况使用。

5. 数据存储模块的实现

数据存储模块是整个系统的数据入口，首先读取指定的 XML 文件，并抽取 XML 结构与现有映射表进行合并，最后把抽取的数据存储到 HBase 数据库中。在整个物联网系统中，传感器传回来的数据会放在一个特定的上传目录中，数据存储模块需要定时扫描这个目录，将新产生的 XML 文档读取并解析存储在数据库中。

如图 7.14 所示为系统物理环境图，数据接收服务器接收来自物联网络的 XML 数据，并存放在特定位置，定时将新产生的 XML 文档打包上传到云端服务器集群，而基于 HBase 数据库的海量 XML 数据存储和检索平台则部署在云端服务器集群上。因此，从整个数据存储模块来看，需要向外提供用来上传 XML 文档的接口。

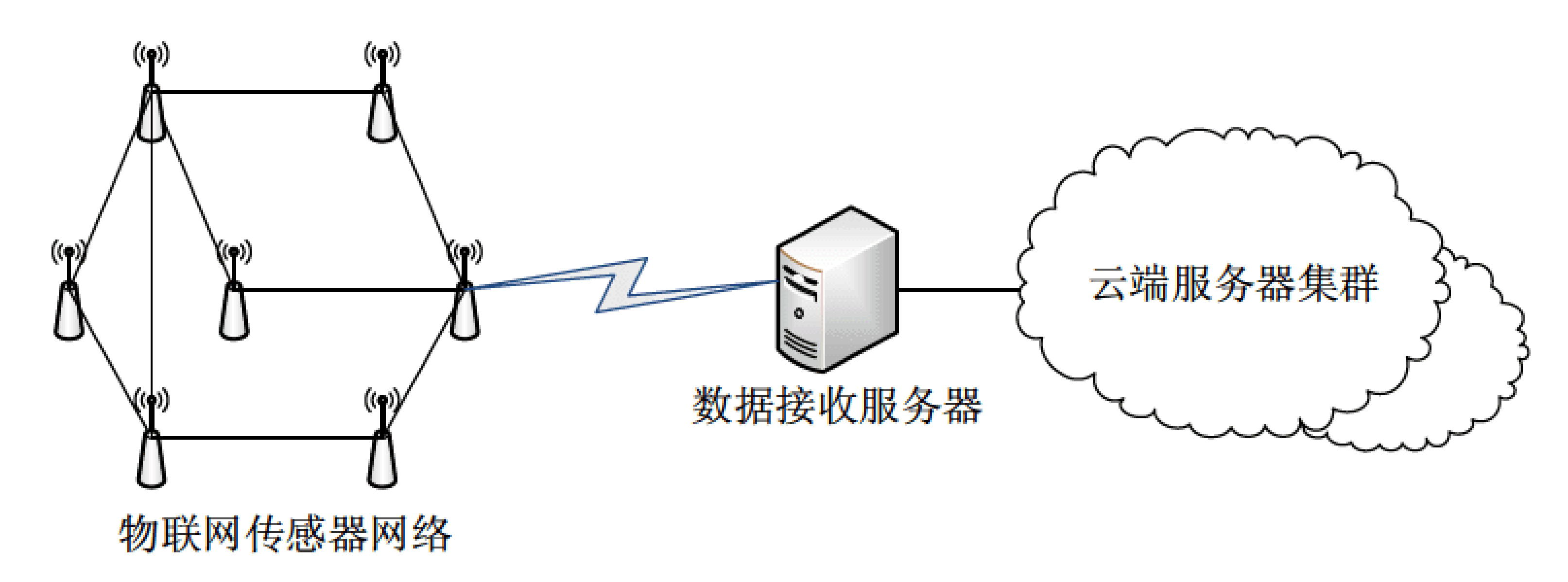


图 7.14　系统物理环境图

（1）XML 结构抽取

XML 数据上传到 HDFS 服务器后，会根据 XML 文件的文件节点首先对 XML 进行分类，由于具有不同文档节点的 XML 文件在存储上属于不同的 HBase 表，因此，将 XML 文件按照文档节点区分类别有利于后续处理。另外，按照文档节点区分后的文档可以使用 MapReduce 并行处理抽取结构，将相同文档节点的 XML 文件 Map 到同一个计算节点上，不同文档节点的 XML 文件之间不会相互产生影响。经过 Map 任务分发后，各个节点执行结构抽取、节点编码和数据存储，最终将 XML 文档存储在 HBase 中。由于存储任务直接在各个计算节点完成，因此不需要对 Reduce 进行特别处理，此处使用一个 Reduce 来统计各个节点完成的存储结果信息，如图 7.15 所示。

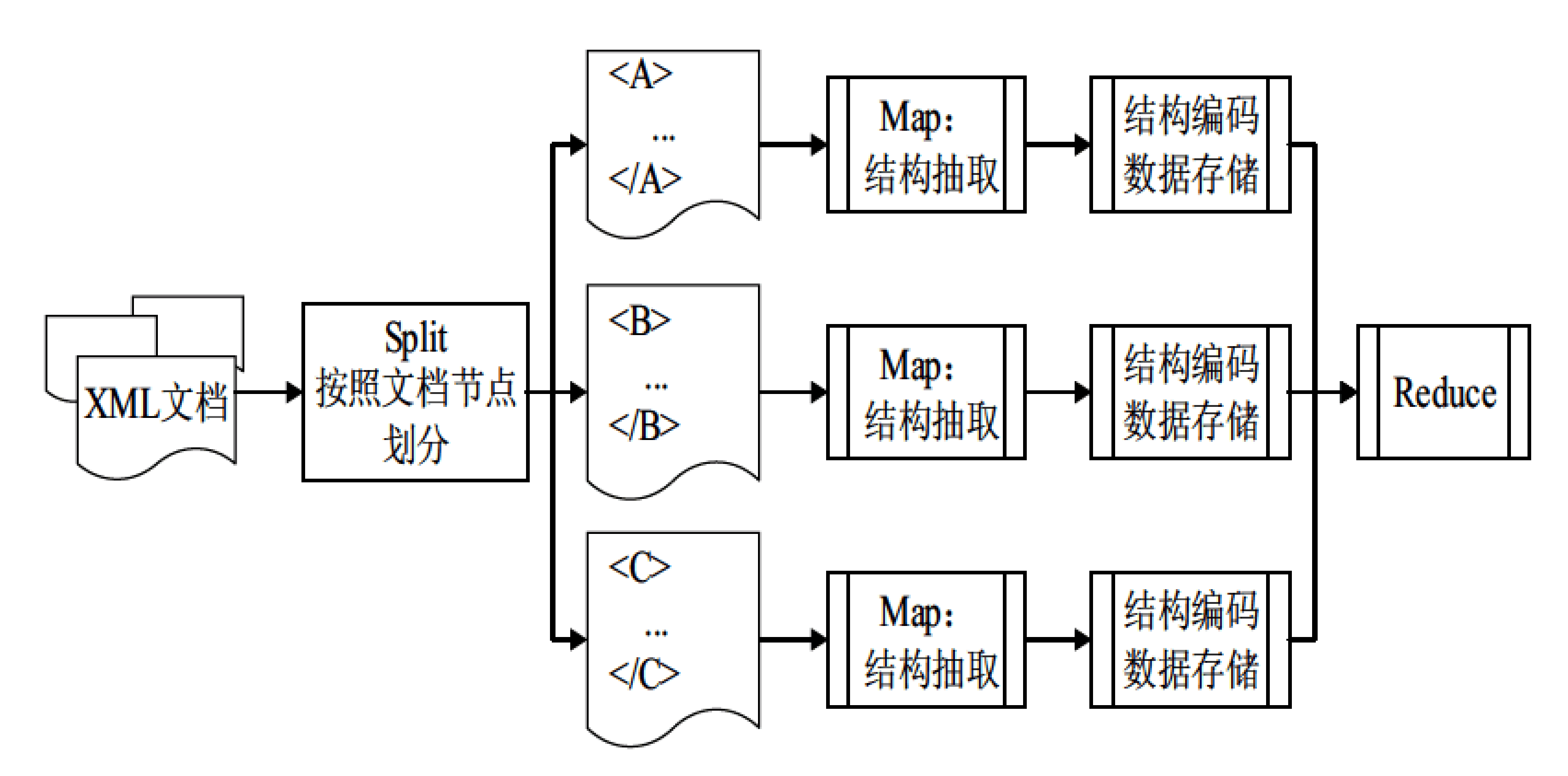


图 7.15　XML 数据存储模块的 MapReduce 流程

XML 的结构抽取事实上是对 XML 文档的遍历过程，无论采用 DOM 或者是 SAX 都可以完成 XML 的结构抽取。由于本项目处理的数据是海量的小 XML 文档，因此使用 DOM 即可，不必担心由于读取大 XML 文件导致内存不足等问题。

（2）XML 节点编码

XML 经过 DOM 解析成树结构，就可以遍历整棵 DOM 树来对文档中的节点编码。整个数据存储过程的流程图如图 7.16 所示。

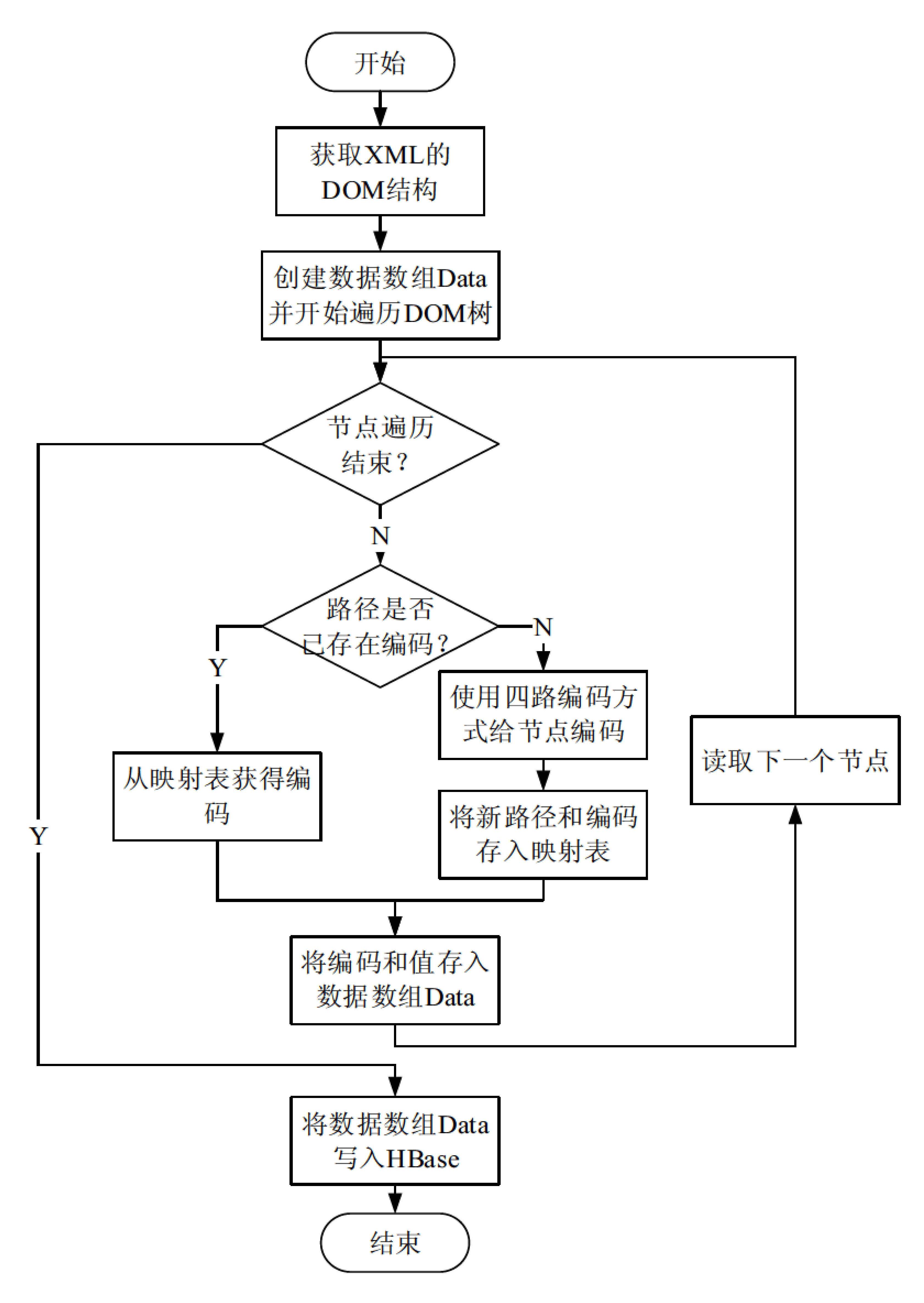


图 7.16　XML 文件存储流程图

首先获取 XML 的 DOM 结构，这一步通过 DOM 解析器得到。之后创建一个数据数组 Data，

这个数组用来存储 HBase 的列名以及其对应的值。一个 XML 文件在 HBase 表中存为一行数据，因此遍历这颗 DOM 树将所有列和值都存在 Data 数据中后，再一次性调用 HBase 数据存储接口将这行数据存入 HBase 中。

DOM 树的遍历是一个 DFS 深度优先遍历的过程，小文件的特性保证了 DFS 的深度不会非常深，因此无需考虑内存溢出问题。对于每一个节点而言，首先通过节点路径查询映射表，看该节点的路径是否已经存在，如果已经存在，那么直接从映射表中获取路径对应的列编码；如果节点路径还没有出现过，映射表中不存在这样的路径，那么需要先通过四路编码方式对节点路径进行编码，并将路径与其对应的列编码保存在映射表中。之后将列编码与对应的节点文本值保存到 Data 数组中。直到遍历结束，将整个 Data 数组写入到 HBase 中的一行。

Data 数据数组是一个二维数组，分别存储着列编号和 XML 节点的值。如果每次得到一个 XML 路径对应的列名，都把 XML 路径的值直接存入 HBase 的相应列，会导致非常多的写入数据库操作，这会直接导致系统写入性能降低。并且，处理完整个文件再写入并不会导致数据一致性问题，即使中途由于某种原因导致系统崩溃，也只是重新解析一次崩溃前正在处理的文件。

在编码节点会产生一个映射表，这个映射表是一个非常重要的数据结构，在后续处理新的 XML 文件的存储或者数据的检索中都作为关键部分存在，能够辅助快速完成存储和检索。

如图 7.17 是映射表的静态类图。MappingTable 是映射表的类，其成员变量是两个 Map 型的结构，其中 p2cMap 是从 XML 的路径到 HBase 列名的映射，而 c2pMap 是从 HBase 列名到 XML 的映射。之前已经提到过，由于 XML 允许在同一个节点下存在名称相同的节点，因此一个 XML 路径可能对应着多个 HBase 中的列，因此 p2cMap 是一个从 String 型到 List<String>型的一对多的映射。列名到 XML 路径是一对一的，因此 c2pMap 是一对一映射。MappingTableManager 是对 MappingTable 的封装，由于系统需要同时处理一些具有不同文档节点的 XML 文档，因此内存中需要保存着多个 MappingTable 的实例。MappingTableManager 中的成员变量就是一个 String 到 MappingTable 的映射，可以通过文档节点查询映射表是否存在，并提供获取或删除指定的 MappingTable 的成员函数。

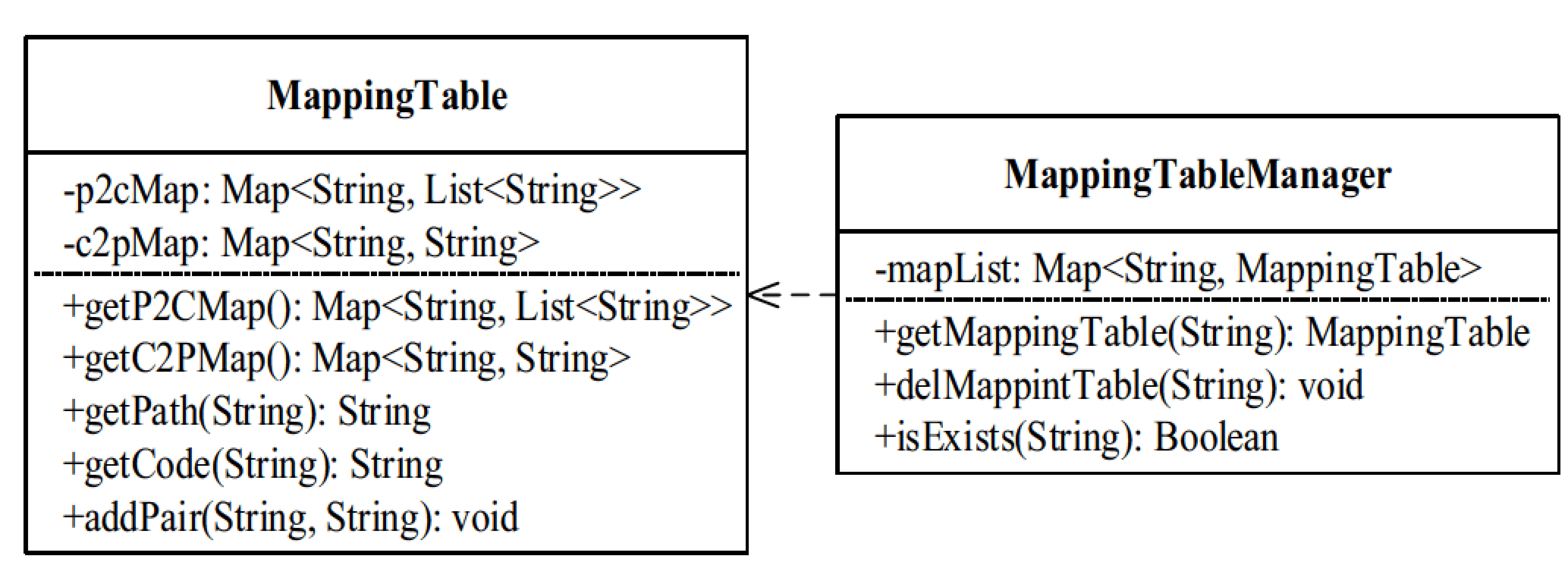


图 7.17　映射表的静态类图

在节点编码流程中，最重要的一步是按照四路编码方式确定一个 XML 路径对应的 HBase 列编码。

算法 4-1 是 XML 文档的编码算法。DecodePath()函数的输入是一个 DOM 节点类型的当前节点 ptrNode，一个 String 类型的当前节点的编号，还有目前该文档节点对应的映射表。该编码算法的主要功能是将一个新输入的 XML 文档按照其结构，给 XML 的所有节点进行编码，如果有新加入的节点，则同时按照四路编码方式给新加入的节点编码，并用新的编码更新映射表。

由于四路编码算法要求知道当前节点是相同父节点的所有节点中的第几个，因此，DecodePath()中由辅助数据结构和函数来完成节点次序相关的记录和查询工作。nameMap 是一个 HashMap，在遍历 ptrNode 的孩子节点时，用来统计当前被遍历到的节点是同名节点的第几个；第 9 行中的 Count()函数是统计映射表中已存在的同名节点的数目；第 20 行的 IsLastNode()判断当前节点是否是映射表中的最后一个节点等。此外，还有很多对编码进行操作的函数，例如第 10 行的 GetCode()函数是获取映射表中以 currCode 为父节点的并且路径为 childNode 的第 currNum 个节点的编号；第 13 行的 ChangeLast()函数是将传入的路径的最后一个数值加上一个值，例如 ChangeLast(“1.5.13”, -1)得到的结果是“1.5.12”；21 行的 LastNum()函数式获取传入的编号的最后一个数值。此外，InsertBewteenNodes()函数是将当前节点插入到两个节点中间，并返回插入后所在位置的编码，事实上这就是四路编码中新节点插入两个节点之间的函数。这些方法的具体实现限于篇幅，不在此一一列举，不过由于都是查映射表就能得到的操作，因此每个操作的最高时间复杂度不会超过四路编码树的宽度，最坏情况下，N 个节点的映射表，其时间复杂度不超过 O(N)。

```
    输入：当前节点 ptrNode，当前节点编号 currCode，映射表 mTable
    输出：更新后的映射表 mTable
    1:   DecodePath(Element ptrNode, String currCode, MappingTable mTable)
    2:   BEGIN
    3:       currPath ← ptrNode 的 DOM 节点路径
    4:       新建 nameMap，类型为 Map<String, int>，记录节点名出线的次数
    5:       FOR EACH childNode of ptrNode DO
    6:       BEGIN
    7:           IF nameMap.get(childNode.name) ≠ Φ THEN BEGIN
    8:               currNum ← ++nameMap.get(childNode.name)
    9:               IF Count(currCode, childNode, mTable) ≥ currNum THEN
    10:                  childCode ← GetCode(currCode, childNode, currNum, mTable)
    11:              ELSE BEGIN
    12:                  firstNodeCode ← GetCode(currCode, childNode, 1, mTable)
    13:                  secondNodeCode ← ChangeLast(firstNodeCode, -1)
    14:                  IF currNum = 2 THEN childCode = secondNodeCode + ".1"
    15:                  ELSE childCode ← secondNodeCode + '.' + ToString(currNum* 4-3)
    16:              END
    17:          END
    18:          ELSE BEGIN
    19:              nameMap.insert(childNode.name, 1)
    20:              IF IsLastNode(childNode, currCode, mTable) THEN
    21:              childCode ← currCode + '.' + ToString(LastNum(currPath, mTable)
+ 4)
    22:              ELSE childCode ← InsertBewteenNodes(childNode, currCode, mTable)
    23:          END
    24:      mTable.addPair(currPath, childCode)
    25:      DecodePath(childNode, childCode, mTable)
```

对 XML 节点编码的过程是一个 DFS 的过程，算法 4-1 的第 5 行循环遍历当前节点的所有孩子节点，并在 25 行出调用 DecodePath()函数，将当前的孩子节点作为参数传入下一层。因此，此算法在调用时，只需要传入 DOM 解析的 XML 文档的根节点，根节点编号“1”，以及解析 XML 文档前的映射表即可。整个编码过程会遍历 XML 树结构的所有节点，因此读入 XML 文件创建映射表的总体时间复杂度约为。

（3）HBase 数据存储

- HBase 数据库表的设计

数据存储首先要对 HBase 数据库中的表进行设计。HBase 是一个高可靠性的、高性能的、面向列的、可伸缩的分布式存储系统，其底层利用 HDFS 作为文件存储系统。HBase 与典型的关系数据库不同，它采用 KeyValue 的底层存储结构，更适合存储半结构化或者非结构化的数据。

图 7.18 HBase 中存储 XML 数据的表模式。事实上，真正的表模式只有一个列族，这个列族的名称就是 XML 文档标签 *d*，而具体到列标签，是可以动态扩展的，但所有的列标签都来自于映射表中与 XML 路径对应的值。实际应用中，由于动态扩展某一个列会导致除了新增加的行之外其他的行都不包含新增加的列，这在传统的关系数据库中将导致空间浪费。然后在 HBase 中，由于新增加的列只是在相应的列族里面添加一个 KeyValue 键值对，因此其他行都不会存在这个空列，所以不会额外占用空间，这就是面向列存储的 HBase 由于灵活的模式而在空间利用上的优势。

Row Key	Column Family:文档标签*d*				
FileName	1	1.1	1.5	...	1.13.1

图 7.18　数据表模式

XML 数据表中的列标签是直接通过四路节点编码方式得到的编码，而要通过编码得到 XML 路径就必须通过映射表查询，然而映射表一般是在内存中的，虽然速度快，但会导致一些问题，例如宕机后如何恢复映射表。为了保证灾难恢复等功能，需要将映射表也持久化在 HBase 中。映射表主要保存 XML 路径到 HBase 列的映射，由于这个映射表中从 XML 路径到 HBase 是一个一对多的关系，正如/BookStore/Book/Chapter 对应着“1.9.1”、“1.9.5”和“1.13”三个列。不过，只要将一个 XML 路径对应的多个列经过一定的序列化就可以存储在同一个单元中。这里采用以“#”符号隔开的方式序列化这些列，例如上面的三个列存储在以/BookStore/Book/Chapter 为列标签的列中，其值为“1.9.1#1.9.5#1.13”。

映射表与数据表不同，由于同一个数据表的所有数据都对应着一个映射表，因此一个数据表的映射表只需要在 HBase 中存一行就可以。这里的映射表在 HBase 中的表名叫 MappingTable。MappingTable 每行的行关键字是这行映射表结构所对应的数据表的表名，因此与 XML 的文档节点名称相同。如图 7.19 为 HBase 存储表的结构图。

行关键字	列族:文档标签*d*			
TableName	Path 1	Path 2	...	Path N

图 7.19　映射表的 HBase 存储表结构

由于结构表也需要持久化，而持久化就会有一致性问题，内存中的映射表和 HBase 中的 MappingTable 如何保证一致性是一个关键问题。如果采用强一致性模型，那么在每次内存中映射表变化后都会导致对 HBase 中数据的更新，这对性能会有比较大的影响；而如果采用弱一致性模型，就必须考虑可能存在的不一致问题。因此，借鉴 HBase 的一些设计思想，这里的系统添加了日志系统来保证结构表的一致性。

结构表采用了预写式日志（Write-Ahead Logging，WAL）。WAL 的核心思想是对数据文件的修改必须只能发生在这些修改已经记录了日志之后。也就是说，如果需要写入数据到 HBase，在写入操作提交到 HBase 之前，应该先将该操作记录于 WAL 日志中。如果遵循这个过程，那么就不需要在每次修改数据后都立即持久化到磁盘，因为即使在出现崩溃的情况时，系统仍然可以用日志来恢复数据库。此外，预写式日志可以减少系统读写硬盘的频率，因为在提交日志的时候，只有 Log 文件需要持久化到磁盘，而不是所有的 HBase 数据内容。此外，由于预写式日志仍然会影响到系统的性能，因此在对于一些写入数据不太频繁并且数据一致性要求不太严格的场景下，可以禁用预写式日志以提高性能。

- 数据存储模块的实现

HBase 存储模块的整体静态类图如图 7.20 所示。

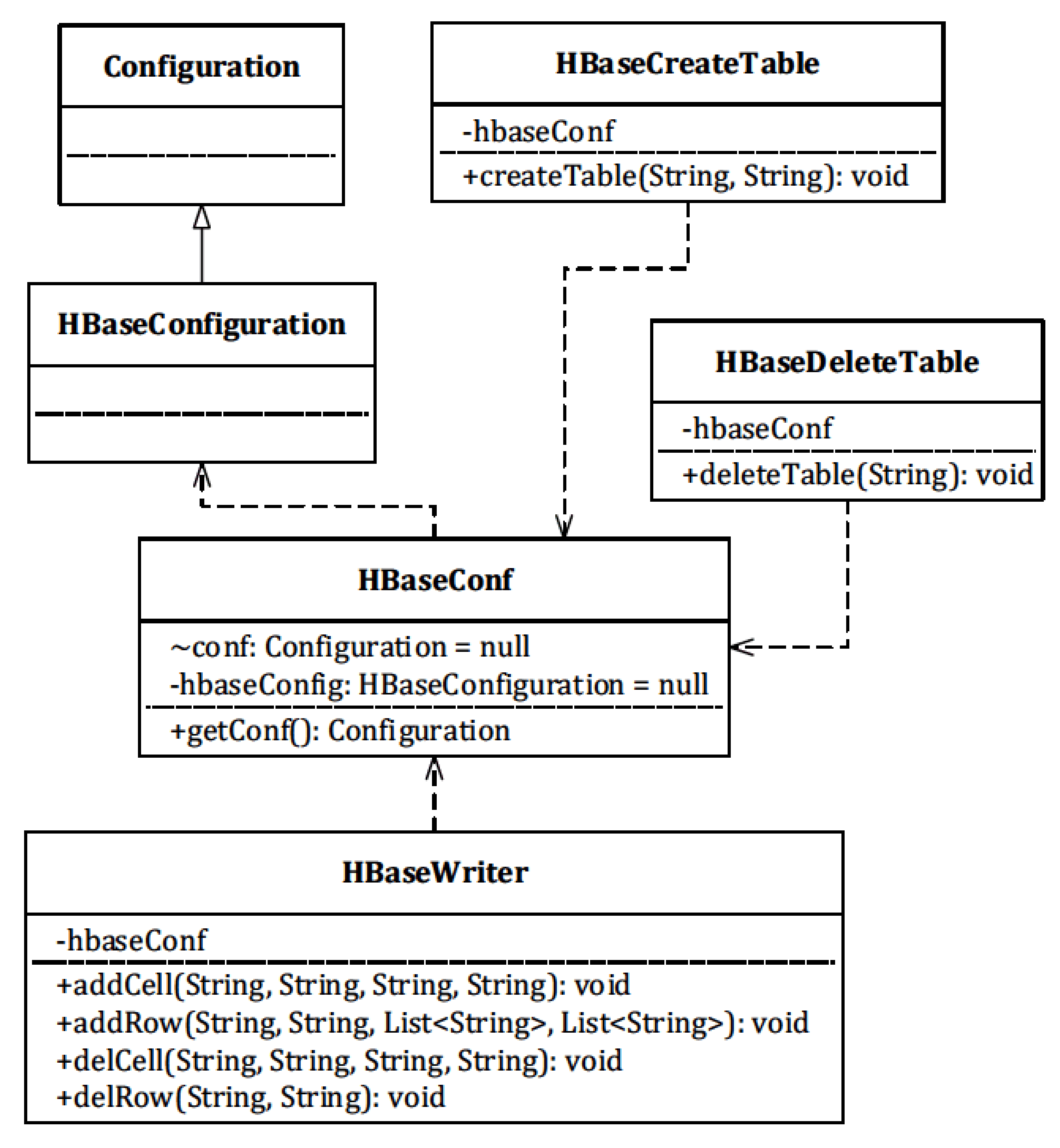


图 7.20　HBase 数据存储模块的静态类图

数据库存储模块主要向上层提供写入数据的接口。这里来自上层的数据主要是写入一行数据或者写入某些特定列数据的接口。由于 HBase API 中提供了对数据写入操作的所有接口，因此此

处只需对原生 API 进行封装，向上层提供统一写入接口和特定功能的写入接口。HBaseWriter 类就提供了写入数据和删除数据的接口，用户可以调用 addCell()方法写入一个单元格，或者调用 addRow()方法写入一行数据。

HBaseCreateTable 类提供了创建新表的接口。使用 createTable 方法就可以创建一个新表，而参数则为表名和列族名。这里列族名只有一个 String 型，是因为无论是数据表还是结构表，都只有一个列族。

HBaseDeleteTable 类提供了删除一个已存在表的接口。使用 deleteTable 方法删除指定的表。

HBaseConf 是对 HBase 的配置类，保存了配置信息，并保持一个连接，其他类会使用 HBaseConf 已经创建的配置连接到 HBase。

如图 7.21 是向 HBase 中写入一行数据的 addRow 函数的流程图，函数的参数包括要写入数据库的表名 TableName、输入文件名、列标签数据数组和对应的值数组。如果要插入的数据来自一个新的 XML 文件节点，那么首先会检查数据库中是否已存在同名的表，如果不存在，则会创建一个新的表，并把数据插入到新创建的表中。

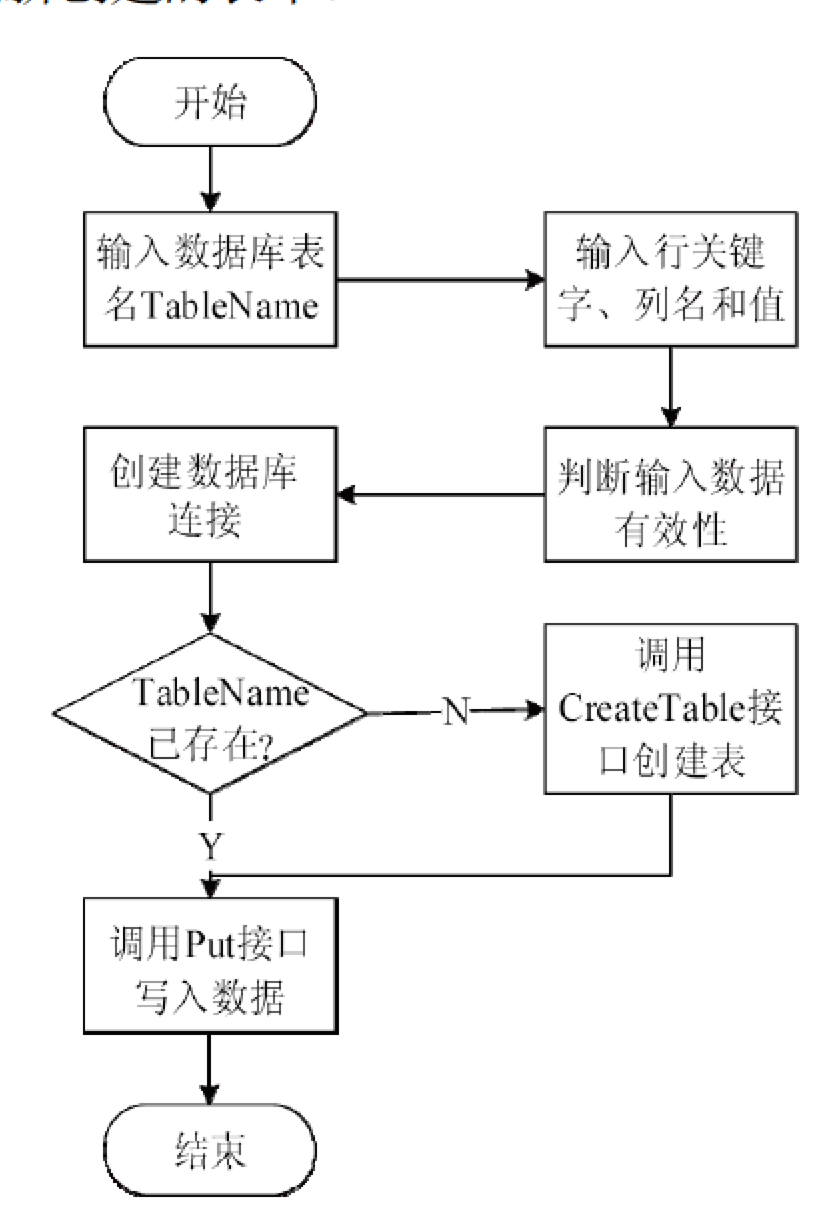


图 7.21　写入数据到 HBase 的 addRow 函数流程图

7.4.2　数据检索模块的详细设计与实现

数据存储映射模型的建立是后续高效数据检索的基础，本小节主要介绍海量 XML 数据存储和检索系统在数据检索方面的设计与实现。数据检索主要是 XQuery 的映射和数据查询两个重要部分。

1. XQuery 查询设计描述

在实现对存储在 HBase 中的海量 XML 文档检索时，我们使用 XQuery 为前端查询语言，XQuery 被设计用来查询 XML 数据，XQuery 相对于 XML 的关系，等同于 SQL 相对于关系数

据库的关系。XQuery 原本的设计是直接基于 XML 文档查询的，由于我们已经把海量小 XML 文档存储在 HBase 中，因此需要将之移植到 HBase 数据库上，就需要修改 XQuery 解析的具体实现。图 7.22 显示了 XQuery 查询的流程。

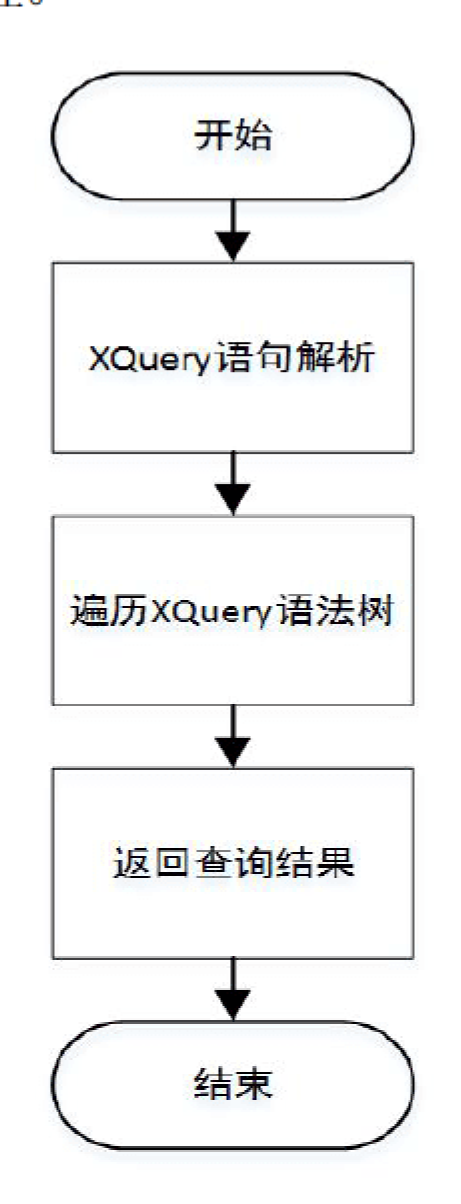


图 7.22　XQuery 查询流程图

XQuery 查询的主要流程有 XQuery 语句解析、遍历 XQuery 语句解析出来的语法树和返回查询结果。在遍历语法树和返回结果的过程中要用到查询结果集，结果集的设计对查询过程有很大影响。

2. XQuery 语句解析

XQuery 语句解析的目标是根据用户输入的语言明白用户查询的意图，分析出语言背后的意义，为下一步具体地查询 HBase 数据库起到指导作用。XQuery 语句解析首先要识别出语句中的标识符（词法分析），标识记号包括：关键字 for、let、where、return、and、or、in 等；符号:=、=、>、<、>=、<=、!=、/等；文字符号包括数字字符串等；以“$”开头后跟字符串的变量。其次要对识别出来的标识符进行语法分析，并构建语法树。在整个语句解析过程中遇到词法错误或者语法错误调用相应的错误处理方法进行异常处理。

针对本系统的 XQuery 语句解析，为了减少工作量，增加系统解析的可靠性和稳定性，我们采用了一个用 Java 开发的最受欢迎的语法分析生成器 JavaCC，这个分析生成器工具可以读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的 Java 程序。JavaCC 还提供 JJTree 工具来帮助我们建立语法树。本文使用 JavaCC 的原因是 W3C 的 XML Query 工作组使用 JavaCC 来构建并测试 XQuery 语法的版本；另一个原因是尽管 JavaCC 不是开放源码，但它是完全免费的。

JavaCC 的输入是巴科斯-诺尔范式（BNF），JavaCC 将 BNF 描述的复杂语言的语法自动生成解析代码。针对 XQuery 语句的 BNF 我们采用 XML 的查询语言的 W3C 规范，W3C 官网上给出的 XQuery 的 FLWOR 语句的 BNF 范式如下。

```
    FLWORExpr ::= (ForClause | LetClause)+ WhereClause? OrderByClause? "return" 
ExprSingle
    ForClause::="for" "$" VarName TypeDeclaration? PositionalVar? "in" ExprSingle ("," 
"$" VarName TypeDeclaration? PositionalVar? "in" ExprSingle)*
    LetClause    ::= "let" "$" VarName TypeDeclaration? ":=" ExprSingle ("," "$" 
VarName TypeDeclaration? ":=" ExprSingle)*
    TypeDeclaration ::= "as" SequenceType
    PositionalVar::= "at" "$" VarName
    WhereClause::= "where" ExprSingle
    OrderByClause    ::= (("order" "by") | ("stable" "order" "by")) OrderSpecList
    OrderSpecList    ::= OrderSpec ("," OrderSpec)*
    OrderSpec    ::= ExprSingle OrderModifier
    OrderModifier    ::= ("ascending" | "descending")? ("empty" ("greatest" | 
"least"))? ("collation" URILiteral)?
```

以上这段 BNF 摘自 W3C 的 XML Query 2002 年 11 月 15 日的工作草案。

通过 JavaCC 和 JJTree 生成的解析代码能够为用户输入的 XQuery 语句生成相应的语法解析树，例如简单的 XQuery 语句：for $a in /bib/book where $a/publisher= “Xidian” return $a/title，经过解析生成图 7.23 的语法树。



图 7.23　语法树

生成语法树后进入下一个流程，遍历生成的语法树，并根据语义进行相应的动作，包括处理变量、构建查询的 XML 路径、查询 HBase 等。

3. 遍历语法树

在正式介绍遍历语法树前有必要对结果集做一定的描述。结果集对整个查询过程很重要，要求结果集能够应对查询内部的递归使用，而且结果集要尽量简洁高效。本系统的结果集中的属性如下所示。

```
String xpath="";
List<String> columns;
List<String> file;
int validNum;
```

xpath 用于存放本结果集的 XML 文档全路径。columns 用于存放查询结果数据在 HBase 中的编码，即 Path2Code 表中的值，也是 Code2Value 表的列名，file 是对应于 columns 编码在 Code2Value 表中的 RowKey（即 XML 文档名）。根据 columns.get(i) 和 file.get(i)即可确定结果集中的具体的一条记录。validNum 用于存放本结果集中的具体结果数。

遍历 XQuery 解析过程中产生的语法树，根据相应的语义完成具体的操作动作。下面以 FLWOR 语句为代表简要介绍遍历语法树过程中的语义动作。

for 语句：记录变量名称并构建变量对应的结果集。代码如下：

```
SimpleNode variable = (SimpleNode) forNode.jjtGetChild(0);
SimpleNode value = (SimpleNode) forNode.jjtGetChild(1);
ResultList forLoopVariableValue = eval( value );
String varName = ((SimpleNode) variable.jjtGetChild( 0 )).getText();
int variableId = m_vars.newVariable( varName, forLoopVariableValue, FOR_INDEX );
```

varName 存放具体的变量名称，m_vars 将变量和其对应结果集绑定在一起，并标记该变量类型是 FOR_INDEX。

let 语句：记录变量名称并构建变量对应的结果集。代码如下：

```
SimpleNode variable = (SimpleNode) letNode.jjtGetChild(0);
SimpleNode value = (SimpleNode) letNode.jjtGetChild(1);
String varName = ((SimpleNode) variable.jjtGetChild( 0 )).getText();
m_vars.newVariable( varName, eval( value ), LET_INDEX );
```

处理过程和 for 语句类似，但是构造结果集操作不同，for 语句直接通过全路径 xpath 获得所有的值，而 let 语句分为两种情况：let 语句中有 FOR_INDEX 类型变量，则根据变量的条件构造结果集，完成对 xpath 范围的指定；没有 FOR_INDEX 类型变量，构造过程和 for 语句相同。

根据生成的 XQuery 语法树，可以判断 where 条件是否为空，并依据此将查询分为两类：

第一类 XQuery 表达式语法树的 where 条件为空，也就是说 Xquery 语句中不包括对节点具体

值的查询，只是找到符合某种路径条件的元素节点集合，例如：

```
for $a in /bookstore/book
return $a
```

第二类则包含了某些需要查询节点具体的值来确定节点是否符合条件的路径，例如：

```
for $a in /bookstore/book
where $a/@id=123
return $a
```

对于第一类查询，在 XML2HBase 映射模型得到映射表后，直接在 XML 文档结构映射表上就能快速得到需要查询的 HBase 列。但第二种查询则需要将查询数据库的部分单独列出来进行处理。

4. 执行查询操作并返回查询结果

对于第一种查询，直接对映射表进行查询的速度非常快。从 path2code 数据映射表中查找相应条件下的所有满足条件的列编码，然后可直接查询 code2value 映射表得到相应的查询结果。

对于第二种查询，需要根据条件从 HBase 中取出满足条件的结果的 columns 和 file。进行语义动作过程如下：

- 根据结果集中的 xpath 属性从 PathToCode 表中获得相应的全路径的编码。
- 以获得的编码为列从 Code2Value 表中取值并判断是否满足条件，若满足条件则分别添加列名和 RowKey 到 columns 和 file 中，在该过程中为了充分发挥 HBase 自身的处理海量数据的特性，系统采用 HBase 的过滤器从 Code2Value 中读取数据。

根据遍历 XQuery 语法树返回结果集中的 columns 和 file 属性，从 HBase 上具体的 Code2Value 表中取得对应的数据，以字符串的形式返回。

- 对于 and 和 or 连接的多条件的 where 语句，对单个条件得到的结果集进行集合的交并操作，and 语句进行“交”操作，or 语句进行“并”操作，由于 columns.get(i)和 file.get(i)确定一条数据，因此结果集的交并操作是针对 columns 和 file 进行的。在进行交并操作前需要将结果集的 columns “同一化”，例如/bib/book/title 和/bib/book/price，对它们的编码值是不同的，它们进行交并操作是无意义的，需要将编码规约到两个路径具有相同父亲元素的层次，对于本例，要从编码中取出/bib/book 的编码，即去掉编码中最后一个“.”符号及其后面的编码，然后进行交并操作，并将结果保存到结果集中。下面为提取父路径的代码:

```
public String getParentCol(String childCol,int n){
    if(n==0){
        return childCol; }
    int offset;
    int offset2=childCol.length();
```

```
        for(int i=0;i<n;i++){
            offset = childCol.lastIndexOf('.', offset2 - 1);
            offset2 = childCol.lastIndexOf('.', offset - 1);}
        return childCol.substring(0, offset2);
    }
```

return 语句：首先构建将要返回的 XML 全路径即结果集中的 xpath，对于没有 where 语句的查询，直接返回结果集；有 where 语句的查询，要根据 where 语句的结果集中的 columns 和 file 获得最终返回结果集中的 columns 和 file。

其实，第二类查询又细分为两类，即直接查询和通过过滤器查询。

（1）直接查询

直接查询是指不需要进行条件判断的查询，例如获取某一列的数据等。这类查询直接使用 HBaseReader 提供的接口即可。此处给出直接查询 HBase 的序列图，如图 7.24 所示。

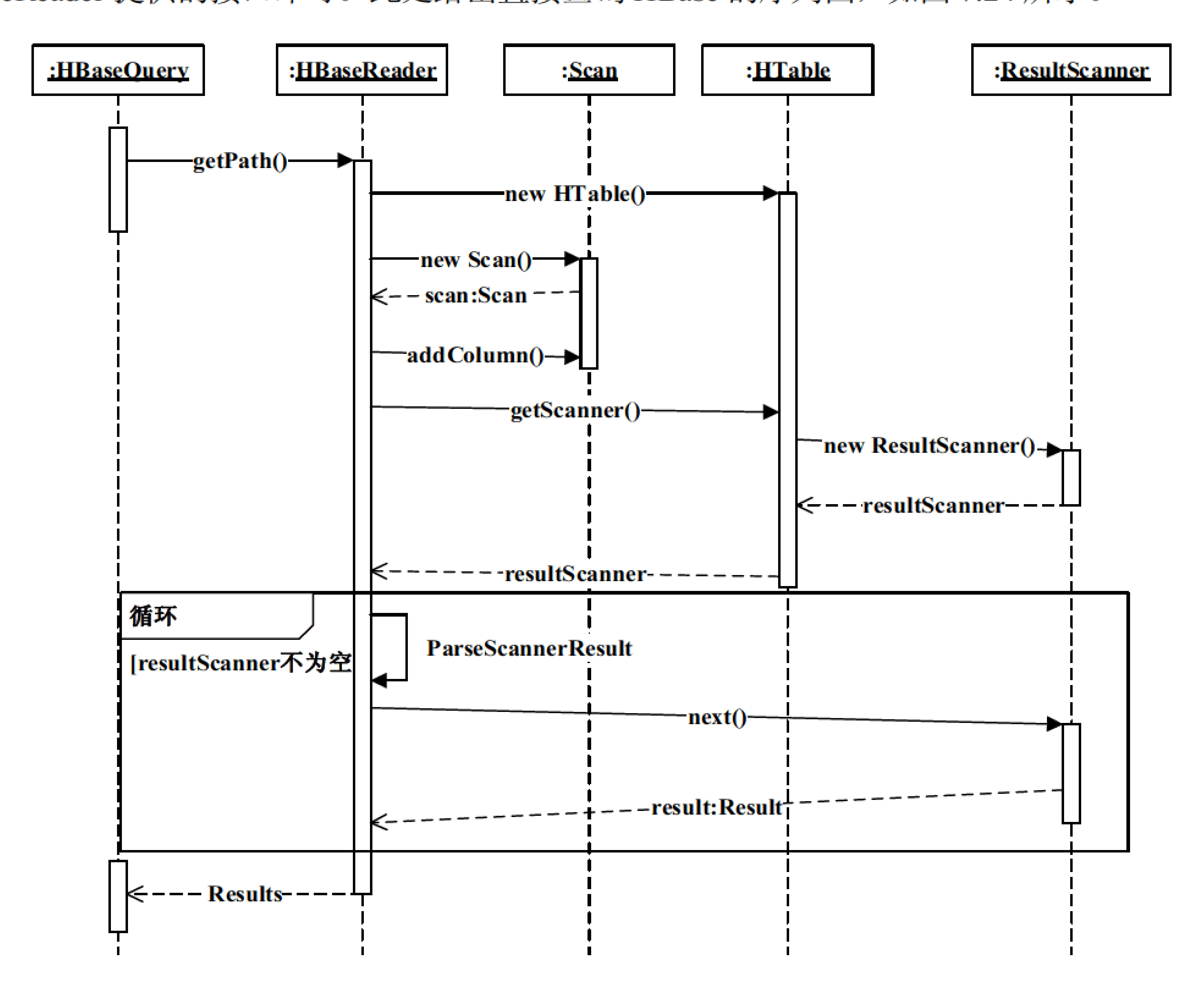


图 7.24　直接查询序列图

步骤 01　调用 query()函数获取指定列，由于调用 HBaseQuery 接口之前已经通过 XML2HBase 映射表获得了要查询的路径对应的列集合，因此 query()函数直接指定为需要查询的列就可以。

步骤 02　调用 HBaseReader 的 getPath()函数，并将 HBaseQuery 实例传过来的传入 getPath()函数。

步骤03　HBaseReader 实例创建一个 HTable 类的实例，对 HBase 中的表进行操作。

步骤04　HBaseReader 实例创建一个 Scan 类的实例, Scan 类是 HBase 表的扫描类, 可以通过 Scan 类的实例指定需要扫描行关键字区间和列等。

步骤05　将需要读取的 HBase 列使用 scan.addColumn()函数添加到 scan 中，并调用 HTable 的 getScanner()方法进行扫描。

步骤06　HTable 实例创建一个 resultScanner 类的对象, resultScanner 类是获取来自 HBase 的结果集，所有的查询结果都保存在 resultScanner 中。

步骤07　HBaseReader 获得 resultScanner 的实例，并解析当前 resultScanner 的当前行结果。

步骤08　使用 resultScanner.next()获取下一行结果，如果 resultScanner 已经没有下一行数据则结束程序循环，否则转到第 7 步。

（2）过滤器查询

关系数据库的 SQL 语句提供了丰富的查询条件，方便用户对表中数据进行过滤。HBase 作为一款数据库，也提供了对数据的过滤功能。HBase 的 Scan 和 Get 操作都能够获取指定行关键字的数据，甚至还可以指定获取特定列族、列标签以及某个时间戳范围内的数据，但对于用户而言，可能还希望对行、列以及值进行过滤，以获取包含特定部分或在符合特定条件的数据，例如获取所有值在 100 到 200 之间的行。

显然，上面的需求可以由客户端通过 Scan 或者 Get 操作获取到数据后，再在客户端编程获取需要的数据，但这种客户端过滤会增加用户实现应用的复杂度，同时占用更多的客户端性能。

事实上，HBase 通过过滤器提供了更细粒度的查询功能，例如通过正则表达式过滤行关键字或者值。HBase 提供了一个过滤器结构类 Filter，并预先实现了多种过滤器。用户需要对数据进行过滤时，可以选择 HBase 已经实现的过滤器在服务端对数据进行过滤。即使 HBase 提供的过滤器功能不够理想，开发者也可以通过实现 FilterBase 类编写自己的过滤器，这使得 HBase 的过滤功能非常灵活。

HBase 提供了丰富的过滤器，图 7.25 中列出了部分 HBase 提供的过滤器，例如 SingleColumnValueFilter 对单个列的值进行过滤，过滤条件可以是字符串比较或者数值比较等。CompareFilter 提供了对包括行关键字、列族、列标签以及值等粒度的条件过滤。此外，根据由于 HBase 提供了 FilterBase 作为过滤器的基类，因此这里可以根据实际情况自己创建特殊的过滤器。

对于本系统，由于数据都是以字节的形式存储在 HBase 中，HBase 自带的过滤器只能对字符串条件进行过滤，而 XQuery 语句支持整数和小数条件查询，因此需要扩充 HBase 的过滤器使其支持整数和小数的过滤操作。下面列出整数过滤器的核心代码，小数过滤器与之类同，将自定义过滤器打包成 jar 文件上传到 HBase 集群中的每个机器上并且重启 HBase 服务器，才能生效。

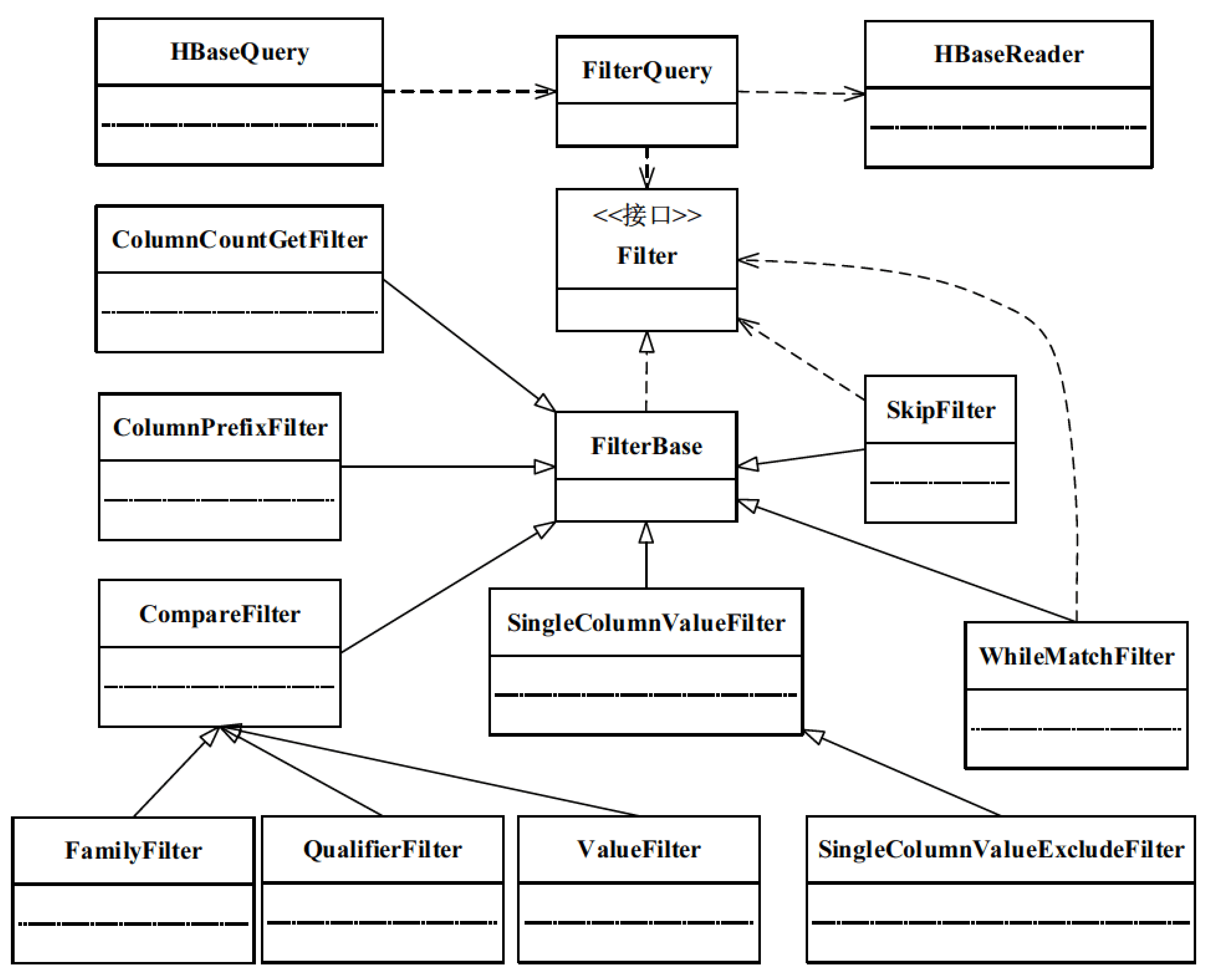


图 7.25　数据检索模块的过滤器查询类图

自定义整数比较过滤器代码如下：

```
private boolean filterColumnValue(final byte [] data, final int offset,
       final int length) {
    int value=Integer.parseInt(Bytes.toString(this.comparator.getValue()));
    int real=Integer.parseInt(Bytes.toString(data, offset, length));
    switch (this.compareOp) {
    case LESS:
      return value > real;
    case LESS_OR_EQUAL:
      return value >= real;
    case EQUAL:
      return value == real;
    case NOT_EQUAL:
      return value != real;
    case GREATER_OR_EQUAL:
      return value <= real;
    case GREATER:
      return value < real;
```

```
        default:
          throw new RuntimeException("Unknown Compare op " + compareOp.name());
        }
      }
```

使用过滤器的查询方式的具体步骤是：

步骤 01　HBaseQuery 类提供条件查询接口，并将具体查询条件发送到 FilterQuery 类的实例。

步骤 02　FilterQuery 根据条件选择适当的 Filter，并将过滤条件填入 Filter 生成具体过滤器实例。实际使用中，条件可能不唯一，因此需要选择多个 Filter。HBase 提供了对多个 Filter 实例的支持，并能够指定查询命中的条件是满足所有的过滤器还是满足其中之一。这些过滤器实例以及过滤器间的过滤条件构成了一个过滤器队列 FilterList。

步骤 03　FilterQuery 将过滤器序列 FilterList 传递给 HBaseReader 实例，HBaseReader 中的 queryWithFilter()方法将 FilterList 作为构造参数传递给 Scan，构成一个 Scan 实例。

步骤 04　创建 HTable 的实例，并调用 getScanner()方法，使用 Scan 实例对 HBase 进行查询，并得到最后的结果集 ResultScanner 实例。

步骤 05　循环解析 ResultScanner 实例中的每一行数据，得到最终查询结果。

5. XQuery 分页查询

由于存储的 XML 文档数据是海量的，因而有可能造成查询出的数据也是海量的，这就极有可能造成查询的过程中内存溢出；即使没有发生内存溢出，随着存储的数据量不断增加，一次查询有可能会消耗很长时间才能有响应（返回查询结果），造成用户体验不佳，面对上面的两个问题，这里采用了分页查询的策略。同时 HBase 的 scan 操作可以通过设置读取数据表的起始行和终止行分页读取数据，该操作为分页查询提供了基础。

XQuery 分页具体实施方案为，对每个 XML 数据表，按一定的步长取记录存储在 HBase Code2Value 表中的 RowKey（即 XML 文档名）作为分页点，然后通过设置 scan 的起始行和终止行实现分页查询，经过多次实验测试，当前系统取步长为 5000 效果较好。由于从一个存储了海量 XML 文档的表中取出所有的分页点是耗时、耗资源的任务，一个存储了 23 万个 XML 文档的表获取所有分页点的时间大概是 12 秒，且一个 XML 数据表的分页点不经常变化，因此每次查询从表中获取一次分页点是不明智的，这里的解决方法是将一个表的所有分页点存放到 HBase 上的 tables 表中的 index 列，tables 的表结构如表 7.7 所示。

表 7.7　tables 表结构

RowKey	Info			
	creater	time	count	index
xmark-10.0-1	admin	2013-04-25	230001	[file0,file1,…filen]
xmark-1.0-0	admin	2013-04-25	20000	[f1,f2,…,fn]

从表 7.7 可以看出 tables 表中除了有每个表的分页点信息外，还有表的创建者、创建时间、

表内的文档数信息。这里维护 XML 数据表 index 的策略是当用户向 XML 数据表中上传 XML 文档树大于等于 5000 时才重新获取一次该表的所有分页点，并更新相应的 index 值。当用户上传少量文件时由于 HBase 存储数据按照 RowKey 排序插入，上传的每个文件都会插入到某两个分页点之间，因此查询时不会缺少刚上传文件的结果。同时可以设置一个后台服务进程，每隔一段时间对所有 XML 数据表进行更新 index 操作。

有了 XML 数据表的所有分页点，另一个好处是可以并行地对多个页进行查询，从而提高整体的查询效率。在本系统中我们采用线程池进行多个页面的并行查询。

```
    ThreadPoolExecutor threadPool = new ThreadPoolExecutor(10, 15, 2,TimeUnit.SECONDS,
new LinkedBlockingQueue<Runnable>());
```

为了减少线程池初期创建线程的时间，初始化最少线程数设置为 10，为了防止过多的线程造成资源消耗过大，设置最大线程数为 15，线程所允许的空闲时间设为 2 秒。

6. XQuery 查询模块的核心类图

XQuery 查询模块的核心类图如图 7.26 所示。

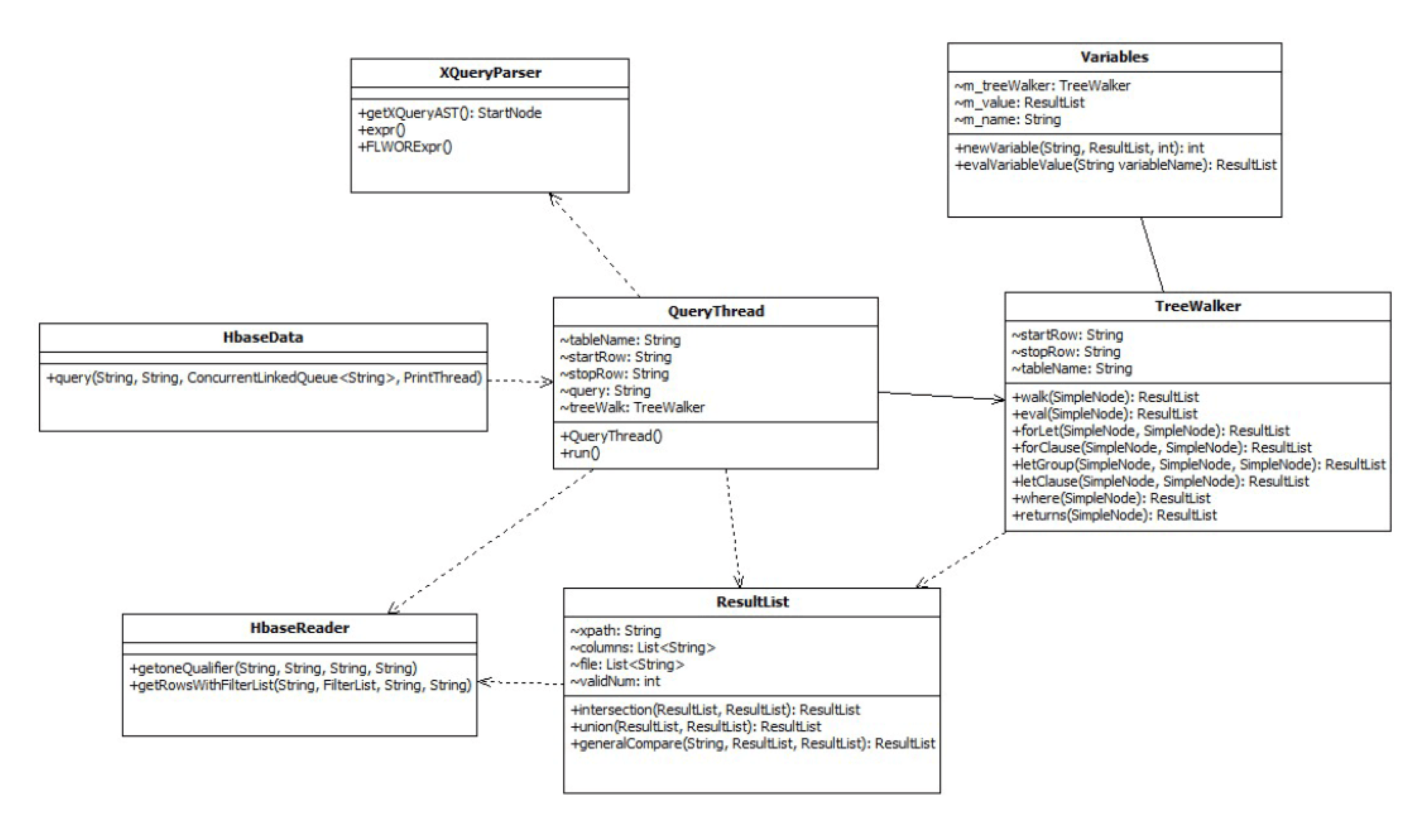


图 7.26　XQuery 查询模块核心类图

从图 7.26 可以看出，XQuery 查询模块核心类包括 HBaseData 类、QueryThread 类、TreeWalker 类、XQueryParser 类、Variables 类、HBaseReader 类和 ResultList 类。HBaseData 类是查询的主入口类，query 函数使用线程池进行查询。QueryThread 类是具体查询的线程类。XQueryParser 类进行 XQuery 语句的解析，最后生成语法树，TreeWalker 类主要遍历语法树，调用 Variables 进行变量操作，ResultList 类是结果集类，包含结果集的交并等操作，HBaseReader 类主要处理从 HBase

中读取数据。

图 7.27 是 XQuery 查询的序列图，从图中可以看出 XQuery 查询的序列为：

步骤01 调用 HBaseData 的 query 函数，首先取出 XML 数据表的所有分页点，创建线程池，为每个查询页运行一个 QueryThread 线程进行查询操作。

步骤02 QueryThread 创建 XQueryParser 对 XQuery 语句进行解析，并构建语法树。

步骤03 QueryThread 创建 TreeWalker 实例，并调用 TreeWalker 的 walk()方法遍历语法树。

步骤04 遍历语法树的过程中创建中间结果集，调用 HBaseReader 中的方法填充结果集。

步骤05 将遍历完成后的结果集返回到 QueryThread 中。

步骤06 QueryThread 调用 HBaseReader 中的方法根据结果集获取查询结果。

步骤07 QueryThread 返回查询结果。

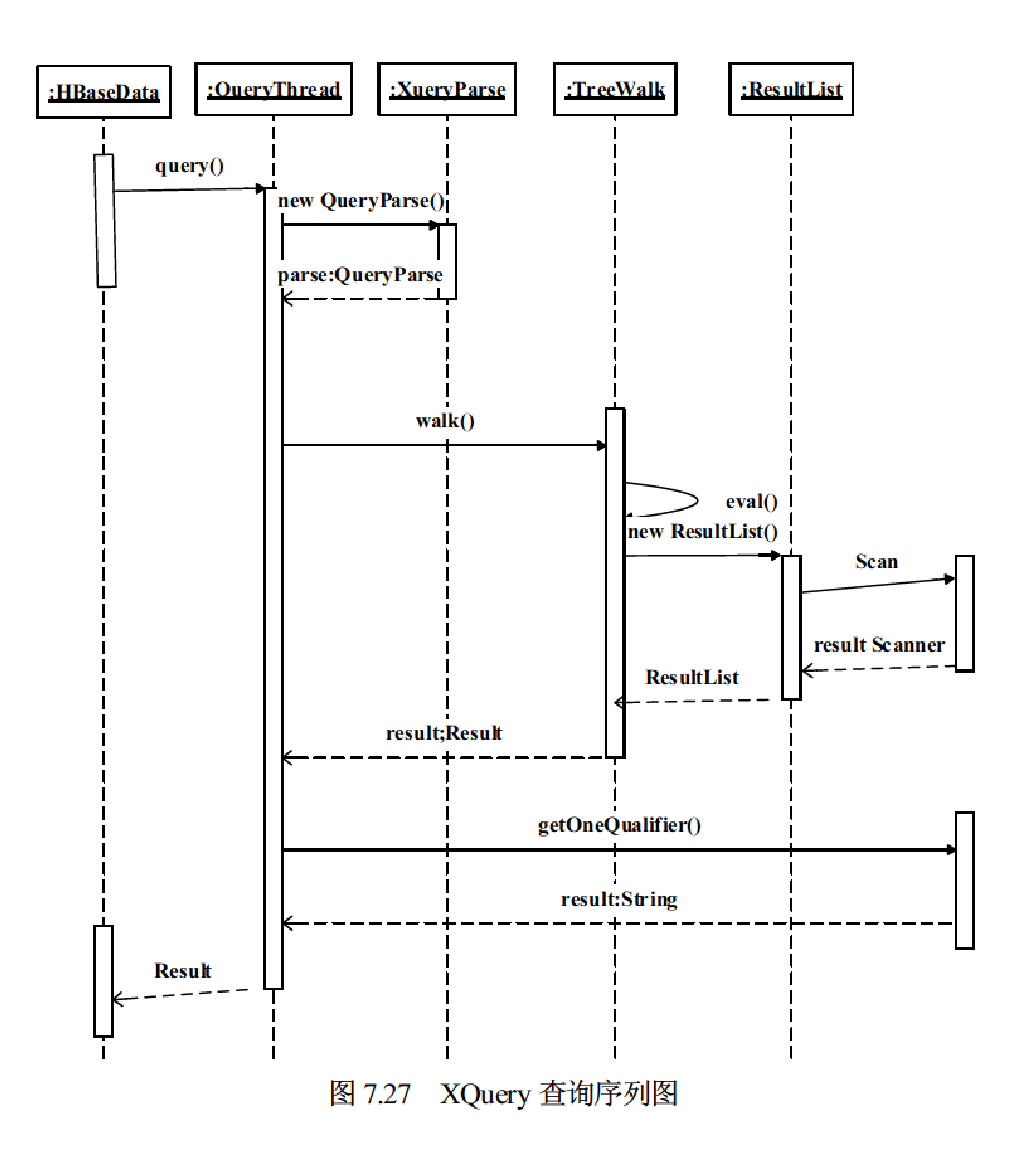


图 7.27　XQuery 查询序列图

7.4.3　用户模块的详细设计与实现

1. 用户模块实现的描述

用户模块主要是对系统进行多用户支持，提高系统的利用率，同时预防某些用户的恶意操作。

与用户模块有关的操作有用户登录、管理员管理用户、管理用户权限等操作。系统为了完成以上的功能或操作需要在服务器端保存用户的用户名、密码、针对每个 XML 数据表的权限等信息，当用户登录或者对用户管理时对 HBase 上的数据进行查找或修改。

记录用户信息的表是 users 表，具体的表格式如表 7.8 所示。

表 7.8 users 表格式

RowKey	user			tables		
	username	password	isAdmin	table1	table2	…..
admin	admin	19901124	yes	2	2	…..
legend	legend	1234	no	1	1	…..

从表 7.8 可以看出 users 表有两个列族，一个列族为 user 保存用户的基本信息，包括用户名、密码和是否为管理员；另一个列族为 tables，该列族保存该用户针对每个 XML 数据表的权限，列名就是 XML 数据表的名称，用户对 XML 数据表的权限分为不可见、可读和可读写三种权限，分别用 0、1、2 代表。每添加一个 XML 数据表需要向 tables 列族中添加一列，并设置相应的用户权限。

用户登录时，需要查询 users 表的 user 列，验证用户名和密码，并查询是否为管理员；管理员添加和删除用户时，添加或删除相应的行，编辑权限时修改 tables 列族的相应列的值。

2. 用户模块的具体实现

图 7.28 是用户模块的核心类图。

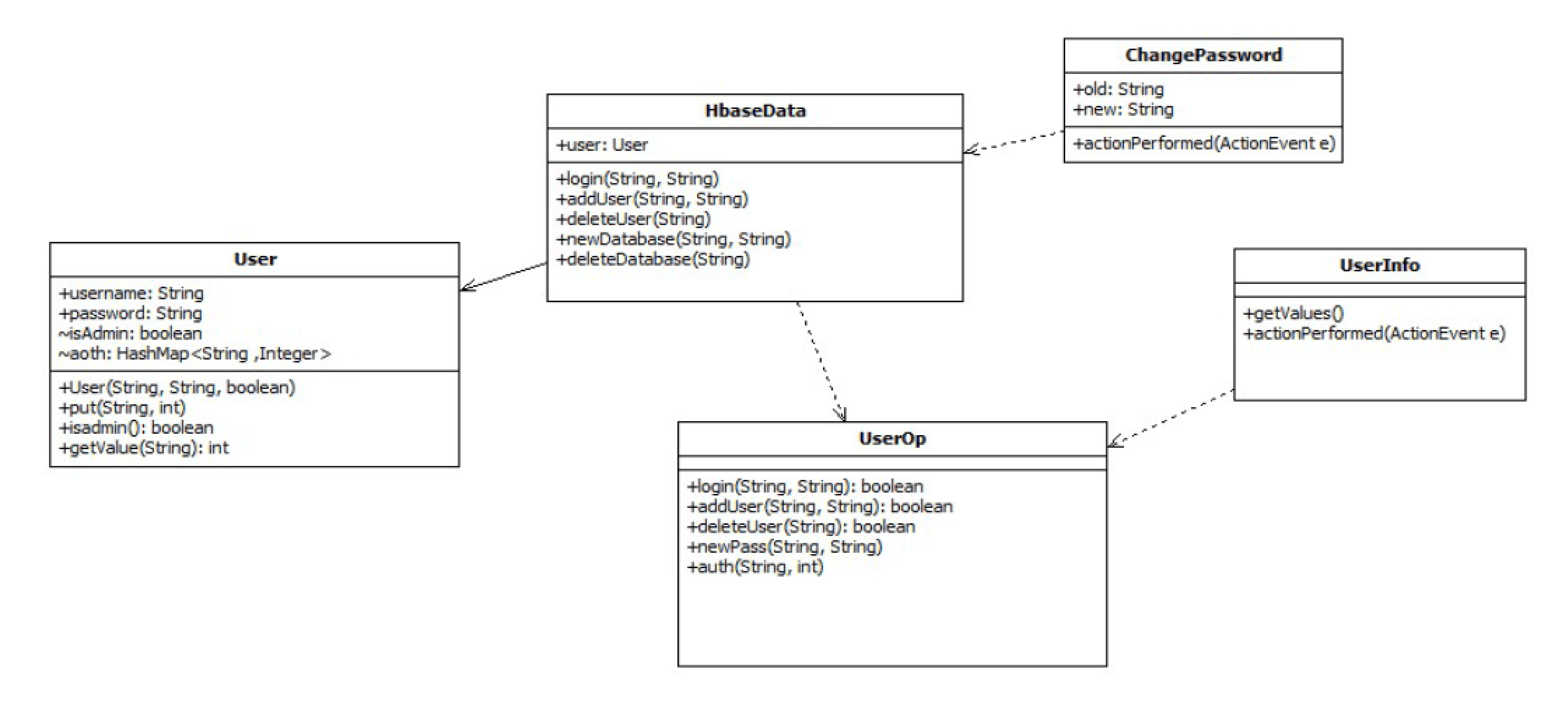


图 7.28 用户模块核心类图

从图 7.35 的用户模块核心类图可以看出，用户模块的核心类包括 User 类、HBaseData 类、UserOp 类、ChangePassword 类和 UserInfo 类。User 类是用户的实体类，User 内的属性和 HBase 的 users 表信息一一对应，User 的方法 getValue(String)获得针对某个 XML 数据表的权限，isAdmin()

返回该用户是否是管理员。UserOp 类是与 HBase 中 users 表的交互类，该类提供了查询修改用户信息的方法，例如 login(String,String)方法用于登录验证，auth(String,int)方法用于修改针对某个数据表的权限。HBaseData 类是用户模块的上层类，该类中包含添加、删除用户，添加、删除数据表等与用户有关的操作。ChangePassword 类是用户修改密码的类。UserInfo 类是为管理员列出某用户的所有权限，并提供权限修改。

至此，海量小型 XML 数据的存储和检索平台已经建立起来，对于本章初提出的问题已成功解决。

7.5 本章小结

随着互联网技术的快速发展，大数据处理成为云计算、物联网领域的一个颠覆性的技术变革。XML 作为一种重要的信息交付格式，对 XML 的存储和检索技术研究已经有很多年的历史了，并且目前也有一些基于分布式的 XML 文件存储和检索的研究。但截至目前为止，仍然没有一个完整成熟的分布式海量 XML 处理系统的解决方案，因此难以满足日益突出的海量 XML 数据存储和检索的需求。针对以上问题，本章以实际项目为背景，借鉴了当前流行的云计算平台 Hadoop 以及分布式存储系统 HBase，利用分布式处理框架，在 HBase 上实现了一个海量 XML 数据存储和检索的平台。本着提高海量数据处理的高可扩展性和高效的目标，这里实现的系统一定程度上解决了海量 XML 数据，特别是海量 XML 小文件的存储和检索问题。

本章主要介绍了以下内容：

- 提出了一个由 XML 数据存储到 HBase 数据库的数据映射机制，采用四路编码算法对 XML 节点进行编码，可以实现从 XML 节点到 HBase 列的双向映射，并能够动态适应 XML 文档的节点增加。
- 提出并实现了一个针对 HBase 的 XML 检索平台，针对不同的检索类型实现不同的检索方式，并使用 MapReduce 并行计算实现高效的检索效率。
- 实现了一个 HBase 数据访问接口，充分地利用了 HBase 数据库的特性。利用 HBase 的分布式特性，实现了具有较好可扩展性和并发性能的数据检索系统。
- 以 XQuery 为前端查询语言，通过对 XQuery 语句的解析并结合 HBase 的操作，实现了对存储在 HBase 中的海量 XML 文档的检索，并且提供对查询结果保存到本地文件的功能。
- 添加用户管理，管理员可以进行添加和删除用户的操作，同时可以针对每个数据表对用户进行权限的管理。

本章详细介绍了海量小型 XML 数据存储与检索系统从无到有、从需求分析到实现的整个过程，希望让读者充分了解利用云平台处理大数据的方式，有利于读者深入理解利用云平台解决大数据的问题。

第 8 章

采用Map/Reduce进行大规模社交网络社团发现

近年来，随着互联网的飞速发展，特别是社交网络的日益普及，为了能够在可接受的计算成本下对网络中蕴含的规律进行研究发现，就需要设法简化研究对象。伴随着信息技术的发展，人们在互联网上的活跃度越来越高，能够接触的信息也越来越多。互联网上的数据以海量的规模存在，并持续高速增长。怎样从海量的数据中挖掘出价值含量高的信息是人们迫切关心的技术。因而，社团发现这一课题越来越受关注。然而网络的规模和结构日益复杂和庞大，现有的一些社团发现算法已不再适用，存在着社团发现结果不稳定、无法发现重叠社团和无法处理大规模网络等问题，特别是许多算法在处理大规模网络时效率很低。聚类算法作为一种非监督学习的方法，是包括数据挖掘、机器学习、模式识别、图像分析等诸多领域数据统计分析的一种常用技术，已经得到科研人员的深入研究。但是，传统的串行式的聚类算法存在着两个问题，已经难以满足实际应用的需求：一是聚类的速度不快，效率不是很高；二是在面对规模比较大的数据时，受制于内存容量的限制，往往不能有效地运行。

针对现有一些算法无法处理大规模网络的问题，本章在随机游走算法（Random Movement Strategy）和仿射传播聚类算法（Affinity Propagation Clustering Algorithm，下文简称 AP 聚类）的基础上，借助 MapReduce 编程模式，将两种算法迁移到云平台上，分别给出随机游走算法和仿射传播聚类算法的并行化。对比两种并行化算法与其他算法在多组测试集上的社团划分质量和运行效果，特别的，在处理大规模网络测试集时，算法的 MapReduce 版本也表现出了较为理想的社团划分质量和运行效果。

8.1 研究背景

近年来，随着互联网的发展，网络的规模越来越大，所包含的结点数往往能达到亿级甚至十亿，在其他领域，如生物领域的细胞数目，所牵涉到的结点数目往往更大，要想探究这些网络的规律，若直接对包含了海量结点及连接关系的网络进行处理，往往需要庞大的计算量，以现有的计算资源，难以满足如此之高的计算需求。而网络内部所包含的结点，以及这些结点之间的连接

关系，往往不是杂乱无章的，而是蕴含了某种规律，我们遇到的大多数网络，其内部都包含社团结构（Community Structure）。所谓社团结构，一般定性的定义为处于社团内部的结点之间连接紧密，而处于不同社团之间的结点连接则相对稀疏。挖掘网络中隐藏的社团结构，对我们研究网络的规律和特性有着重要的参考作用。

社团结构在许多方面具有重要意义和作用，例如市场营销、犯罪团伙发现、病毒传播研究等。移动通信运营公司可以在社团的基础上利用个性化推荐来为用户推荐业务，因为社会网络中一个社团有可能是兴趣爱好相近的人群，这样可以提高推荐的成功率。在生物学网络中，社团往往代表某种特定功能的模块，这种社团结构有助于人们理解其运行机理，进而能更清晰地探察其内部逻辑，发现未知的规律。在计算机科学中，对社团的研究和分析有助于在并行计算中合理分配任务以减少各个计算节点之间的通信成本，从而提高计算效率。总之，社团结构能帮助人们直观地认识复杂网络的结构和功能，从而可以更好地理解和利用复杂网络。

正因为社团结构对人们在理解复杂网络的结构和功能方面的重要性，许多研究者投入了对社团发现算法的研究，并提出了许多新的社团发现算法。经过最近十来年的发展，社团发现的研究取得了重要进展，并在很多领域有了成功的应用。划分网络中社团结构的算法可分为 4 大类。

1. 第一类是基于图论的算法

基于图论的算法基本思想是给定一个网络，将其分解成一些子网络，各个子网络内的节点数基本相等，并且处于不同子网内的节点之间的连接非常少。图分割（Graph Partitioning）是计算机科学中的方法。基于图分割的著名算法有 Kernighan-Lin 算法（简称为 K-L 算法）、基于拉普拉斯图特征值的谱平分法和 W-H 快速谱分割法等，其中，K-L 算法的主要思想是先随机或者根据已知信息将原图划分成两个规模已知的子图，在之后每一步的迭代步骤中，为了获得品质因数 Q（定义为表示子图内边数跟子图间边数的差别大小）的最优解，在两个子图中选择规模相等的子图，进行交换，在迭代过程中为了避免只达到局部最优的 Q 值，会允许 Q 值有下降，最终达到一个稳定的最优解。K-L 算法最大缺陷是必须为算法预先指定两个社区的大小，否则算法会得到错误的划分结果，这就使 K-L 算法的应用非常有限，在大多数的真实网络中根本无法得到应用。另一个常用的图分割算法为谱平分法，其主要思想是利用了拉普拉斯矩阵的谱特性，该拉普拉斯矩阵 L 是一个实对称矩阵。人们在使用这类方法时，预先不能确定究竟将图分成多少个子图才合适，因为谱平分法只能将图分成两个子图，或者说偶数个子图，且不知何时停止。

2. 第二类是基于模块度优化的社团发现算法

基于模块度优化的社团发现算法是目前研究最多的一类算法。其思想是将社团发现问题定义为优化问题，然后搜索目标值最优的社团结构。由 Newman 等首先提出的模块度 Q 值是目前使用最广泛的优化目标，该指标通过比较真实网络中各社团的边密度和随机网络中对应子图的边密度之间的差异来度量社团结构的显著性。模块度优化算法根据社团发现时的计算顺序大致可分为三类。

基于模块度优化的算法中第一类算法采用聚合思想，自底向上进行，典型代表算法有 Newman 快速算法和 CNM 算法等。Newman 快速算法将每个节点看作是一个社团，每次迭代选择产生最

大 Q 值的两个社团合并，直至整个网络融合成一个社团。整个过程可表示成一个树状图，从中选择 Q 值最大的层次划分得到最终的社团结构。该算法的总体时间复杂度为 O(m(m+n))。CNM 算法是基于 Newman 快速算法，并采用了数据结构“堆”来对网络的模块化度进行计算和更新，其复杂度只有 $O\left(n\log^2 n\right)$。在很多不同的现实网络中，凝聚算法的确已经得到了广泛的应用，但这并不能掩饰这类算法所存在的问题。首先，在一些应用中，即使已经知道了社区数目，却并没有得到正确的社区结构。其次，凝聚算法倾向于找到社区的核心，而忽略社区的周边。

基于模块度优化的算法中第二类算法主要采用分裂的思想，自顶向下进行。例如，Newman 最早提出的 GN 算法就属于这类算法，算法通过依次删去网络中边介数（即网络中经过每条边的最短路径数）最大的边，直至每个节点单独退化为社团，然后从整个删边过程中选取对应最大 Q 值时的结果。该算法复杂度较高，为 $O(n^3)$。GN 算法尽管有很大的突破，但是仍然有着它自身的缺陷，即对于网络的社区结构，它并没有在量方面给出合理的定义。也就是说，它不能直接从网络的拓扑结构判断它所求的社区是否合理，即不能断定所得到的社区就是实际网络中的社区结构，因此经常需要一些额外的信息来辅助判断所得社区结构是否具有实际的意义。此外，在社团数目不清楚的情况下，GN 算法同样不知道分解应该进行到哪步停止才合理。

基于模块度优化的算法中第三类算法是模拟退火法，模拟退火法被用在很多领域，它的基本思想是将解空间分为若干个状态，定义一个全局函数，通过状态的转换，来获得全局函数值的增量，或者负增量（也称为噪声），通过若干步骤的转换，全局函数获得一个全局最优值，其中噪声的引入是为了避免全局函数只取得局部最优解。应用在社团发现中，它的具体实现中，状态的转换分为局部转换和全局转换，局部转换指的是将某个结点转移到另一个社团，全局转换指的是两个社团的合并或者将一个社团分成多个社团。一种标准的方法是同时结合了局部转换和全局转换的方法，模拟退火方法计算复杂度较高，但获得的社团划分结果较准确，适合规模较小的网络。

总的来说，模块度优化算法是目前应用最为广泛的一类算法，但是在具体分析中，很难确定一种合理的优化目标，使得分析结果难以反映真实的社团结构，尤其是分析大规模复杂网络时，搜索空间非常大，使得许多模块度近似优化算法的结果变得更不可靠。

3. 第三类是动态方法

动态方法主要分为三类：自旋模型法、随机行走法和同步法。1984 年 Fu 和 Anderson 就提出了利用自旋模型分割图的方法，Reichardt 将自旋玻璃的 Potts 模型引入到了确定网络社团最优划分的问题上，通过构造与网络连接相关的 Hamilton 函数，把社团划分与寻找系统的基态结合起来形成自旋模型法；随机行走算法由 Hughes 提出，主要思想是从某个结点出发，下一个结点的选择是随机的。随机行走算法被用在社团发现中，是由于社团内部连接相对紧密，从而随机行走的路径更多在社团内部。从而可以用随机行走的规则来发现社团；一个由相互作用的结点组成的网络中，属于同一社团的结点之间由于连接紧密，相互作用较强，因而在状态更替中，能够更快地达到同步状态，因此可以利用这个性质，通过观察结点的状态同步状况，来反推出网络中的社团结构情况形成同步法。

4. 第四类是重叠社团发现算法

重叠社团发现更符合真实世界的社团组织规律，成为近几年社团发现研究的新热点，涌现出许多新颖算法。重叠社团发现算法大致可以分为三类。

重叠社团发现算法中第一类是基于团渗透改进的重叠社团发现算法，由 Palla 等提出的团渗透算法是首个能够发现重叠社团的算法，该类算法认为社团是由一系列相互可达的 k-团（即大小为 k 的完全子图）组成的，即 k-社团。算法通过合并相邻的 k-团来实现社团发现，而那些处于多个 k-社团中的节点即是社团的“重叠”部分。

重叠社团发现算法中的第二类是基于模糊聚类的重叠社团发现算法，可将重叠社团发现归于传统模糊聚类问题加以解决，以计算节点到社团的模糊隶属度来揭示节点的社团关系。这类算法通常从构建节点距离出发，再结合传统模糊聚类求解隶属度矩阵。值得一提的是，此类算法的关键在于所构建的距离矩阵，采用何种节点距离更符合实际情况在具体应用中是一个值得探索的问题。

重叠社团发现算法中第三类是基于边聚类的重叠社团发现，以往社团发现算法的研究均以节点为对象，考虑如何通过划分、聚类、优化等技术将节点归为重叠或不重叠的社团。Evans 等和 Ahn 等分别发表了以边为研究对象来划分社团的文献。虽然节点属于多重社团，但边通常只对应某一特定类型的交互（真实网络中的某种性质或功能）。因此，以边为对象使得划分的结果更能真实地反映节点在复杂网络中的角色或功能。

但许多社团发现算法存在复杂度较高，无法处理大规模网络结构，社团发现结果不稳定等问题。本章正是围绕上述问题展开，实现了两种可并行化的社团发现方法，使并行化的 RMS 算法和并行化的 AP 算法能够有效处理大规模网络，并且社团划分质量较好。

8.2 相关理论和技术

8.2.1 社团结构

通常，复杂网络中的社区由相似属性的个体组成，或者说它们在网络中担任了相近的角色。现实社会中存在大量关于这种群体的实际样例，例如家庭、同事、班级同学、朋友等，这些群体在某种意义上可以看作是社区。互联网的飞速发展使得各种互联网应用层出不穷，特别是 Web 2.0 的兴起，在这些互联网应用中许多具有相同爱好或者具有相似属性（如同学关系）的用户会形成特定的群体，体现出“人以群分”的特点，这些互联网中的群体与传统的群体相比甚至超越了地域的限制，更具包容性和开放性。 社区具有内部联系紧密、社区之间联系相对稀疏的特征。

当前普遍认为复杂网络中的社区结构是重叠的，即网络中存在同时属于多个社区的节点，这就是所谓的重叠社区。由于人们往往具有多种兴趣爱好，很多用户会属于多个社区。再比如每个人具有多种社会属性，因此而属于多个不同的群体，最普遍的是同学群体、同事群体、家庭群体

等，这也是重叠社区的一种体现。

8.2.2 相关社团发现算法

1. 派系过滤算法

派系过滤算法（Clique Percolation Method，CPM）是比较早的可以发现重叠社区结构的社区发现算法，由 Gergely Palla 等人在 2005 年提出。Gergely Palla 等人认为社区可以看做是由一些相互连通的完全子图（即 clique 或称派系）组成的集合。k-clique 表示完全子图的节点数目为 k。如果两个 k-clique 有 k-1 个公共节点，则称这两个 k-clique 是相邻的。

通过 k-clique 的定义可以知道，在一个 k-clique 中任意选择 $t(0<t\leqslant k)$个节点，这 t 个节点也会形成一个完全子图。所以在 CPM 算法中只需要寻找各个最大的完全子图，之后可以通过构造一个 clique 之间的矩阵，再根据参数 k 和这个矩阵来寻找相应 clique 产生的连通子图。

CPM 算法的主要过程如下：

（1）找出网络中的 clique。

（2）根据找出的所有 clique 构造一个由 clique 之间的关系形成的矩阵（clique-clique overlap matrix）。该矩阵的每一行和每一列对应一个 clique，矩阵非对角线上的元素表示对应两个 clique 的公共节点数目，对角线上的元素表示对应 clique 的节点数目。因此该矩阵是对称的。

（3）根据参数 k，将第（2）步获得的矩阵中对角线上元素小于 k，非对角线上元素小于 k-1 的这些项设置为 0，其他元素设置为 1。

（4）CPM 算法认为社区是由连通的 k-clique 组成的子图。根据第（3）步处理后的矩阵，对角线为 1 对应的 clique 表示满足条件的 clique，非对角线为 1 表示相应两个 clique 的相邻关系，因此通过这个矩阵可以方便地得到各个连通部分，也就是各个社区。

派系过滤算法对 clique 相邻关系的要求很高，在许多网络上的社区发现结果模块化系数并不理想。并且派系过滤算法的复杂度较高，在处理规模较大、边较为稠密的网络时效率很低。

2. EAGLE 算法

EAGLE 算法（agglomerativE hierarchieA clusterinG based on maximaL cliquE）是一种基于合并相似极大派系的层次重叠社区发现算法，由 Shen 等人于 2009 年提出。网络中的一个极大 clique 表示该 clique 不是任何其他 clique 的子集。EAGLE 与 Newman 快速算法类似，也属于凝聚的方法，但是与 Newman 快速算法在一开始每个节点属于单独的社区不同的是，EAGLE 算法是以满足条件的 clique 作为单独的社区。

EAGLE 算法的主要过程如下：

（1）找出网络中的所有极大 k-clique。忽略次要极大 clique（subordinate maximal clique，指该 clique 中的节点在一些其他更大的 clique 中已经出现，但是该 clique 并不是其他任何 clique 的子集）。余下的 clique 和次要节点（subordinate vertex，指因舍弃节点数未达到 k 的 clique 而产生

的那些不在任何 clique 当中的孤立节点）都被当作初始社区。并计算每个社区对之间的相似度。

（2）选择相似度最大的两个社区进行合并，并计算这个新社区和其他社区之间的相似度。

（3）重复步骤（2）直到只剩下一个社区。

EAGLE 算法的复杂度很高，第一过程（合并社区）的时间复杂度为 O（$n^2+(h+n)s$），其中 n 是网络的节点数，s 是极大 clique 的数目，h 是相邻的极大 clique 对的数量。第二过程（从树状图中选择一层作为社区结果）的时间复杂度为 O（n^2s）。并且还要算上找出网络中所有极大 clique 的时间。因此 EAGLE 难以在大规模的真实网络上应用。

3. GCE 算法

GCE 算法（Greedy Clique Expansion）也是一种通过贪心优化局部指标来获得高度重叠的社区结构的社区发现算法，由 Lee 等人于 2010 年提出。

由于 GCE 算法把极大 clique 作为初始社区种子并从其出发进行扩展，不可避免地会出现近似重复的社区，即当前扩展的社区和已经扩展得到的社区基本上是同一个社区。为了判断两个社区是否是重复的，首先定义 δ 函数为两个社区之间的距离：

$$\delta_E(S,S')=1-\frac{|S\cap S'|}{\min(|S|,|S'|)} \qquad \text{式（8-1）}$$

如式（8-1）中，S 和 S' 是两个社区。

给定一组社区的集合 W 和一个社区 S，W 中所有与 S 距离小于 ε（一般取 0.4 能取得较好的结果）的社区都认为是 S 的重复社区。

基于此，GCE 算法的总体过程如下：

（1）找到网络中所有节点数不小于 k 的极大 clique，作为社区种子。

（2）选择没有进行扩展的种子中节点数最多的种子，根据适应函数进行贪心扩展，直到加入任何节点都会降低适应函数的值，此时这个社区记作 C'。

（3）如果已经生成并被接受的社区中存在与 C' 距离小于 ε，则认为 C' 与已有社区重复，因此放弃 C'；否则接受 C'。

（4）返回步骤（2）继续执行，直到所有种子都已被扩展。

GCE 算法做了许多工程上的优化，并且很多参数都是经验值，但是其在人工网络（LFR benchmark）及一些真实网络上效果很好，而且效率很高，是目前较为优秀的重叠社区发现算法。尽管 GCE 算法在实现时做了尽可能的优化，但在处理大规模并且稠密的网络时仍会导致扩展的过程很慢，成为算法效率的瓶颈。

4. LPA 算法

节点所属的社区和其邻接点有很大的关系，一种简单的想法是每个节点按照其邻接点的社区情况来选择所要加入的社区。Raghavan 等人据此提出了一种基于标签传播（label propagation algorithm，LPA）的社区发现算法。

LPA 算法的主要过程如下：

（1）初始化，给每个节点分配一个唯一的标签。对于节点 x，$C_x(0)=x$。

（2）设置 t=1。

（3）将网络中的节点随机排序，假设顺序为 X。

（4）对 X 中的每个节点 x，将 x 的标签设置为新的标签。$C_x(t)=f(C_{x_1}(t),...,C_{x_m},C_{x_{m+1}}(t-1),...C_{x_k}(t-1))$，f 函数返回节点 x 的邻接点中出现次数最多的标签；如果这样的标签存在多个，则从出现次数最多的标签中随机选取一个。

（5）如果每个节点的标签都与其邻接点中出现次数最多的标签相同，则算法结束；否则设置 t=t+1，返回步骤（3）继续执行。

需要指出的是，这里的 f 函数为异步传播方式。异步方式（asynchronous）指当前迭代过程中节点 i 的新标签既和节点 i 的邻接点在上一轮迭代过程中的标签相关（如果这个邻接点该轮迭代过程尚未更新标签），也和节点 i 的邻接点在该迭代过程中的标签相关（如果这个邻接点该论迭代过程已经更新了标签）。同步方式（synchronous）只和上一轮迭代相关。异步方式在处理二部图和星形图时不存在振荡问题，并且异步方式相比同步方式需要更少的迭代次数就能趋向平衡而使算法可以终止，具有更好的性能。

LPA 算法的时间复杂度为 $O(km)$，其中 k 是迭代次数，m 是网络的边数，因此其时间复杂度接近线性，效率很高，能处理很大规模的网络，并且获得的社区结构的模块化系数也较理想。但同时由于 LPA 算法有很多随机性因素，使其结果不能稳定，每次产生的社区结构会存在一定差异。

5. COPRA 算法

Steve 提出了一种扩展的 LPA 算法，能够发现重叠社区结构，即 COPRA 算法（Community Overlap PRopagation Algorithm）。COPRA 算法通过让每个节点可以携带多个标签来支持重叠社区结构，并且针对每个标签有一个隶属系数（belonging coefficient），为了避免传播过程结束所有节点都带有相同的标签集合，在每轮迭代结束后，删除隶属系数小于 1/v（v 是算法的输入参数，表示每个节点最多属于的社区个数）的标签。如果所有隶属系数均小于 1/v，则只保留隶属系数最大的标签，删除所有其他标签；如果存在多个最大的隶属系数，则随机选择一个对应的标签。然后对这些保留的标签对应的隶属系数进行归一化。

因此 COPRA 算法需要指定 v 参数，此参数对社区发现的结果影响很大，一般面对真实网络我们又无从知道每个节点最多属于的社区个数，所以 COPRA 算法有一定局限性。

6. 划分聚类法

给定一个包含 n 个“数据点”的数据集，这些数据点可以是坐标点，或一张网页，或一张图片。把这 n 个数据点划分构造成 k 个分组，每一个分组代表一个聚类。而且 k 个分组满足以下条件:(l)每一个分组至少包含一个数据点；（2）每两个组别之间没有重复的数据点，即每一个数据点属于且仅属于一个分组。划分聚类法在初始阶段要设定好聚类的数目 k，通过不断地一次次迭代优化分组的情况，使得每一次迭代后的分组的结果都比前一次迭代的聚类结果要好。聚类结果好

的标准是：同一个分组中的数据点的相似性越大越好，而不同分组的数据点相似性越小越好。

K-means 法就是划分聚类法的典型代表。K-means 聚类算法思路比较简单，但也是十分实用的一个算法，广泛应用于各学科和工业生产上。K-means 算法早在 1967 年提出，先来描述-下这个简单的算法。K-means 一开始先选择 k 个初始类簇中心（centroid），k 是用户一开始指定的一个参数，也就是用户想要的聚类数目。每次迭代中，每一个数据点都被赋到最近的类簇中心中，等所有的数据点都赋给类簇中心之后，重新更新每一个类簇（cluster）的新的类簇中心。不断地迭代直到没有数据点改变它所属的类簇，或者等价于直到每个类簇中心保持不变。

初始化选择三个点作为类簇中心，然后每一个数据点根据和类簇中心的距离，选择最近的一个赋给类簇中心。这里，我们用坐标的平均值（mean）作为类簇中心。在每一个数据点都赋给类簇中心之后，再更新类簇中心的坐标。下一步，数据点就重新赋给新的类簇中心，类簇中心再重新更新。每一个小图代表其中的一次迭代的过程，K-means 最后结束在迭代 8 次之后，因为类簇中心不再变化，这样聚类结果就出来了，这些聚类结果（clusters）由类簇中心指示着。把数据点赋给与它最近的类簇中心，需要一个衡量相似度的方法来表示“最近”。欧氏距离（Eucliden distance: L_2）是欧氏空间中数据点常用的相似度衡量方法，cosine 距离（cosine distance）则常用于文本图像间的距离计算。当然，除此之外，还有其他几种描述相似性距离的方法，比如曼哈顿距离（Manhattan distance: L_1）也可以描述欧氏空间中数据点的距离，而 Jaccard 方法也常应用于文本的距离计算。

7. 层次聚类法

给定一个数据集，层次聚类法要对这个数据集进行层次性的分解或者层次性的聚合，直到满足某种条件为止。具体又可以分成两类，一类我们称之为“自底向上”，通过慢慢合并类簇达到聚类效果；一类我们称之为“自顶向下”，通过慢慢分裂类簇达到聚类效果。在“自底向上”方法中，初始化时每一个数据点都代表一个类簇，在接下来的迭代中，如果有两个类簇间足够相似，那么，我们就把这两个类簇合成一个类簇，直到所有的类簇不再变化。

怎样判别两个类簇是否足够相似，要给出一个衡量类簇之间距离的方法。定义点 i 和点 j 间的相似值 S_{ij}，类簇 c1 和类簇 c2 之间的相似值为 S(c1,c2)，则计算 S(cl,c2)的简单方法有三种：

（1）两个类簇间数据点的最小相似值。S(cl,c2)=min S_{ij}，$i \in c1, j \in c2$。如果要求两个类簇间的任意两个点都比较相似，用最小相似值比较好，缺点是聚成的类别较多。

（2）两个类簇间数据点的最大相似值。S(cl,c2)=max S_{ij}，$i \in c1, j \in c2$。如果要求聚成的类簇比较集中，类别数目较少，用最大相似值比较好，缺点是聚成的类簇之间可能存在不相关的点。

（3）两个类簇间数据点的平均相似值。$S(c1,c2)=\frac{1}{i*j}\sum S_{i,j}$，$i \in c1, j \in c2$。

8.2.3　Hadoop 分布计算框架

2004 年，Google 公司的 Jeffrey Dean 和 Sanjay Ghemawat 发表了一篇后来产生了重要影响的

关于分布式计算的论文。该论文提出了一种简单有效的分布式计算模型，即 MapReduce。Hadoop 开源项目起源于 2005 年，是 Apache 的一个子项目。Hadoop 定位为可靠的、可扩展的分布式计算框架，目前已有 Hadoop Common、HDFS、MapReduce、Cassandra、HBase 等多个子项目。Hadoop Common 包含其他子项目使用到的公共库。HDFS 是分布式文件系统，用于支持分布式计算环境中的数据存储和读写。MapReduce 即是对 Google 提出的分布式计算模型 MapReduce 的开源实现。目前 Hadoop 已经发展成为一个较为完善的分布式计算框架，并已被广泛使用。

MapReduce 是 Google 的一项重要技术，它是一个编程模型，用于处理海量数据。对于海量数据，通常采用的处理方法是并行计算。对许多开发人员来说，并行计算还是一个比较遥远的概念，并且不易实施。MapReduce 是一种简化了的并行计算的编程模型，它极大地降低了并行计算的门槛，可以让那些没有多少并行计算经验的开发人员开发并行应用。简单地说，MapReduce 是一种简化的分布式编程模型。MapReduce 的运行环境能够解决输入数据的分布细节，自动分布程序到一个由普通机器组成的超大集群上并发执行，并能处理集群中节点的失效，管理集群内部机器之间的通信请求。这样的模型允许开发人员不需要具备详细的并发或者处理分布式系统的经验，就可以处理超大的分布式系统的资源。

在 MapReduce 中，数据由称为 Mapper 的相互隔离的任务进行处理，Mapper 的输出作为另外一个称为 Reducer 的任务的输入，由 Reducer 处理后产生最终结果。MapReduce 计算的输入是一个键值对的数据集合，在 Map 阶段，MapReduce 的运行环境将输入数据集合拆分为大量的数据片段，然后将每一个数据片段分配给一个 Map 任务。MapReduce 运行环境负责调度这些 Map 任务到集群中的各个节点去运行。每个 Map 任务将其分配到的数据集合经过计算产生中间结果，也即对于每一个输入键值对<kl,vl>，Map 任务会调用用户定义的 Map 函数进行处理，产生不同的键值对<k2,v2>的集合。然后，MapReduce 运行环境会对中间结果进行排序处理产生一个新的键值对<k2,v2*>集合，相同键的值被归纳在一起，并将新的键值对集合划分出和 Reduce 任务数量相同的片段数。在 Reduce 过程中，每个 Reduce 任务将分配到的新的键值对<k2,v2*>集合作为输入，对于每一个键值对调用用户定义的 Reduce 函数，并产生新的输出结果键值对<k3,v3>。

Hadoop MapReduee 分布式计算框架属于主从结构。Hadoop 中存在一个作为控制节点的 JobTracker，用于调度和管理其他 TaskTracker。JobTracker 可以运行于集群中任一台计算机上。TaskTracker 负责执行任务，必须运行于 DataNode 上，因此 DataNode 既是数据存储结点，也是计算结点。JobTracker 将 Map 任务和 Reduce 任务分发给空闲的 TaskTracker，让这些任务并行运行，并负责监控任务的运行情况。如果某一个 TaskTracker 出故障了，JobTracker 会将其负责的任务转交给另一个空闲的 TaskTracker 重新运行。

MapReduce 是构建在 Hadoop 分布式文件系统 HDFS 上的大规模数据处理计算框架，基于 MapReduce 模型的应用程序能够运行在上千个商用机器组成的大型集群上，并以一种高可靠、高容错的方式并行处理 T 级别的数据集。MapReduce 计算模型将运行于大规模集群上的并行计算抽象为两个函数 Map 和 Reduce。适合用 MapReduce 模型进行处理的数据集（或任务）有一个基本要求：待处理的数据集可以分解为许多小的数据集，而且每个小数据集都可以完全并行地进行处理。

图 8.1 说明了采用 MapReduce 处理大数据集的过程，首先将大数据集分解为小数据集，每个

（或若干个）数据集分别由集群中的一个结点进行处理（Map）并生成中间结果，框架会对 Map 的输出先进行排序，然后这些中间结果又由大量的结点进行合并（Reduce），形成最终结果。

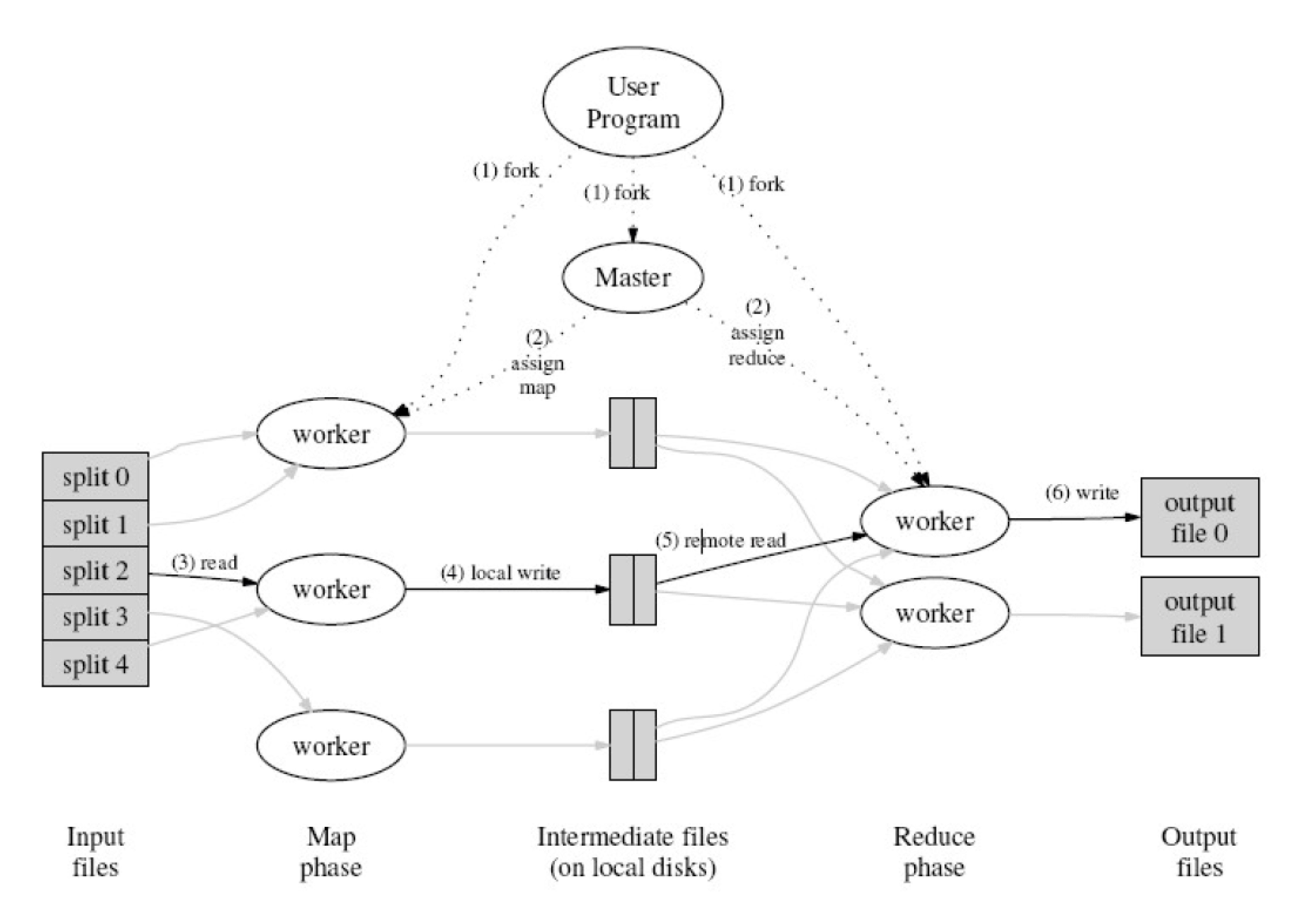


图 8.1　MapRecuce 计算模型

HDFS 的底层实现是把文件切割成块（Block），这些 Block 分散地存储于不同的 DataNode 上，每个 Block 还可以复制数份存储于不同的 DataNode 上，从而达到容错容的目的。

NameNode 是整个 HDFS 的核心，它通过维护一些数据结构，记录每一个文件被切割成了多少个 Block、这些 Block 可以从哪些 DataNode 中获得、各个 DataNode 的状态等重要信息。通常，MapReduce 框架和分布式文件系统 HDFS 是运行在一组相同的节点上的，也就是说，计算节点和数据存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，可以高效利用整个集群的网络带宽。

MapReduce 框架由一个单独的 Master JobTracker 和每个集群节点的 Slave TaskTracker 共同组成。Master 负责调度构成一个作业的所有任务，这些任务分布在不同的 Slave 上，Master 监控它们的执行，重新执行已经失败的任务。而 Slave 仅负责执行由 Master 指派的任务。计算模型的核心是 Map 和 Reduce 两个函数，这两个函数由用户自行实现，其主要功能是按一定的映射规则将输入的<key, value>对转换成另一个或一批<key, value>对输出，如表 8.1 所示。

表 8.1　MapReduce 函数说明

函数	输入	输出	说明
Map	<k1, v1>	List(<k2,v2>)	1. 将小数据集进一步解析成一批 <key,value> 对，输入 Map 函数中进行处理 2. 每一个输入的 <k1,v1> 会输出一批 <k2,v2>。<k2,v2> 是计算的中间结果
Reduce	<k2,List(v2)>	<k3,v3>	输入的中间结果 <k2,List(v2)> 中的 List(v2) 表示是一批属于同一个 k2 的 value

把原始大数据集切割成小数据集时，通常让小数据集小于或等于 HDFS 中一个 Block 的大小（默认是 64MB），这样能够保证一个小数据集位于一台计算机上，便于本地计算。有 M 个小数据集待处理，就启动 M 个 Map 任务（可能小于 M 个，由运行参数决定），注意这 M 个 Map 任务分布于 N 个节点上并行运行，Reduce 任务的数量 R 也可由用户指定。

对 Map 的中间结果先做 Combine，即将中间结果中有相同 key 的<key, value>对合并成一对。Combine 的过程与 Reduce 的过程类似，很多情况下可以直接使用 Reduce 函数。但需要注意的是，Combine 是 Map 任务的一部分，在执行完 Map 函数后立即执行。Combine 能够减少中间结果中<key,value>对的数目，进而减少网络流量。把 Map 任务输出的中间结果（即做完 Combine 后已排好序的结果）做 Partition，按 key 的范围划分成 R 份（R 是预先配置的 Reduce 任务的个数），划分时通常使用哈希函数，如 hash（key）mod R，这样可以保证某一段范围内的 key，一定是由一个 Reduce 任务来处理，可以简化 Reduce 的过程。Map 任务的中间结果在做完 Combine 和 Partition 之后，以文件形式存于本地磁盘。存放中间结果文件的 TaskTracker 会通知主控 JobTracker，JobTracker 再通知 Reduce 任务到哪一个 DataNode 上去取中间结果。注意所有的 Map 任务产生中间结果均按其 key 用同一个 Hash 函数划分成了 R 份，R 个 Reduce 任务各自负责一段 key 区间。每个 Reduce 需要向多个 Map 任务结点取得落在其负责的 key 区间内的中间结果，然后执行 Reduce 函数，形成一个最终的结果文件。

8.3 RMS 算法的并行化实现

8.3.1 RMS 算法

在社会系统、科技系统以及生物系统中，许多个体都是一个紧密相连的团体中的一部分。所谓的紧密团体就是该团体里的个体倾向于经常在一起，并且经常交换信息。例如，一个家庭一般都是住在一起，在一个地方生活；一个实验室的成员常常在一起研究讨论。在这里，我们引入了 agent 集合，集合里的每个元素个体即每个 agent 都代表相应网络中的一个节点，并且可以在一个二维空间中移动。在发现过程中，每个 agent 会在它的所有邻接点中，以一个可变概率随机地挑

选一个邻接点相应的 agent 作为目的地，从自己的坐标点移动到该目的地 agent 所在的坐标点。直观上，我们不难理解，如果节点属于同一个社区，则它们之间的紧密度值应该偏高，即这些节点对应的 agent 之间应该是紧密相连的。很自然地，可以想象这些移动的 agent 在移动的过程中，将逐渐形成一些团体。并且这些紧密相连的 agent 最终将聚集在二维空间的同一个坐标点。

该方法通过网络的局部探索来寻找每个节点所归属的自然社区。在此过程中，不管节点是否已被划分到某个社区，它都会被多次访问到。通过这种方式，重叠社区能很容易地被划分出来。同时通过调节社区层次结构的分辨率，可以发现网络所有层次下的社区结构。

我们将想要进行分析的网络看作是一个简单图 G，其有 n 个顶点和 m 条边。图 G 的邻接矩阵是 A_{n*n}，矩阵中的元素 a_{ij} 代表节点 i 和 j 之间的关系强度。特别地，在无权图中，如果 i 和 j 是邻接点，则 a_{ij} 等于 1；否则 a_{ij} 等于 0。而在加权图中，a_{ij} 的值就等于 i 和 j 之间的权重值。邻接矩阵对角线上的元素无定义，为了方便，均将值设为 0。图中的两个节点是邻接点意味着它们之间有边连接。为了更清楚地表达 RMS 算法，我们使用下标来区分 agent 和其相应的节点，例如，i_0 代表节点 i 对应的 agent。

初始状态下，网络中的每个节点自己形成一个独立的社区，即初始时总共有 n 个社区。此时每个节点对应的 agent 所处的坐标也是不同的。即 n 个节点初始时有 n 个社区，它们对应的 agent 总共也处于 n 个不同的坐标点。网络中的每个 agent 被其他的一些 agent 吸引，然后随机地选择其中的一个 agent 作为目的地，并且以一个随机移动概率从自己的坐标点移动到目的 agent 所在的坐标点。这个概率的值与社区吸引性有关。这个吸引性会最终引导所有的 agent，要么移动到新的坐标点，要么继续呆在原来的坐标点保持不动。最终，这些随机移动的 agent 会趋于一个稳定状态。换言之，大多数的 agent 最终会停留在某个坐标点不再改变，而 agent 在二维空间的分布情况即反映了网络社区最终的划分情况（位于同一坐标点的 agent，它们对应的节点即处于同一社区）。

1. 初始化

给网络中的每个节点定义一个相应的移动 agent，然后根据计算紧密度矩阵 M，初始时每个节点形成一个独立的社区。

将社会网络建模为无向图 $G=(V,E)$，其中 V 表示网络中节点的集合，E 表网络中边的集合，函数 $\tau(v):V\to V$ 表示节点 v 的邻接点集合，$\tau(v)=\{u\mid u\in V\wedge(v,u)\in E\}$ 紧密度矩阵 $M=(m_{ij})_{|V|\times|V|}$ 定义了 V 中节点之间的紧密度值，利用公式（8-2）计算得到节点 i 和节点 j 之间的紧密度值。

$$m_{ij}=1+|\tau(i)\cap\tau(j)| \qquad \text{式（8-2）}$$

为每个节点构造一个初始社区，首先定义初始社区集合 $R_0=\{R_0(1),R_0(2)...\}$，其中 $R_0(i)$ 表示第 i 个社区，初始由节点 i 构成的集合，即 $R_0(i)=\{i\}$，函数 $f(i)\to R_0(i)$，并且 $\forall i\in V$，为 i 指定一个全局唯一的社区编号，代表 i 属于该社区，令 $\mathrm{I}(i)$ 表示 i 的社区编号值。

2. 随机游走过程

执行基于社区紧密度的随机游走策略，需要对 V 中的节点进行遍历并计算，终止条件是“G 中的社区总个数为 1”或“V 中所有的节点的社区编号不再发生变化”。$\forall i\in V$，一次计算的具体过程为：

首先定义 $R^{'}(i)=\{R_0(k)\,|\,j\in\tau(i)\wedge j\in R_0(k)\}$ 表示节点 i 的邻接节点所属的社区集合，计算 $R^{'}(i)$ 中每个社区对 i 的社区吸引性 CA，其中社区 $R_0(k)$ 对 i 的社区吸引性 CA 采用公式（8-3）得到：

$$CA(i,R_0(k))=\left[\frac{\sum_{s\in\tau(i)\cap R_0(k)} m_{is} \sum_{o,p\in\tau(i)\cap R_0(k)} m_{op}}{\sum_{o,p\in R_0(k)} m_{op}}\right]^t \qquad 式（8-3）$$

其中 $t>0$ 是多尺度参数，也称为分辨率参数。调节 t 的取值，可以得到不同的社区划分结构；根据上步得到的 $R^{'}(i)$ 中每个社区对 i 的社区吸引性 CA，计算 i 对 $R^{'}(i)$ 中每个社区的随机移动概率，比较得到的所有随机移动概率，i 从自己所处的社区开始往随机移动概率最大对应的社区移动，更改 i 所属的社区号 $\mathrm{I}(i)$，i 对社区 $R_0(k)$ 的随机移动概率采用公式（8-4）得到：

$$p(i,R_0(k))=\frac{CA(i,R_0(k))}{\sum_{R_0(e)\in R^{'}(i)} CA(i,R_0(e))} \qquad 式（8-4）$$

RMS 算法在以下几个方面不同于其他的社区结构发现算法：

（1）RMS 的概率与 agent 间的社区吸引性（Community Attraction，CA）有关，并且这个概率在随机游走的过程中是可变的，即是自适应的。

（2）传统的分类方法，最大的不足是如果节点在早期被划分进了不合适的社区，那么这些节点将再也没有机会被划分进合适的社区。与这些算法相比，RMS 算法是一个社区发现方面的概率算法。它基于随机游走的 agent，从而当节点在早期不小心被划分到不正确的社区时，仍然有机会在后期离开这个不正确的社区并移动到正确的社区。

（3）对重叠社区的研究转换为了对不稳定节点（其相应的 agent 不是处于固定的坐标点）的分析。因此，选择统计节点在不同社区的频度，并利用节点的社区倾向性概念，来定量地分析社区结构的这种重叠特性。

（4）RMS 仅仅需要考虑每个节点的邻接点。该方法可以很方便地扩展到有向网络（即网络中的边是有方向的）和加权网络（即网络中的每条边有权重）中。

8.3.2 RMS 算法在 MapReduce 上的实现

1. 初始化节点为社区

输入文本为所有节点与邻接点的对应，格式为每一行对应一个节点的邻接信息，由节点 ID 和邻接点 ID 组成，中间使用分隔符分隔。输出文本为输入网络的所有节点的初始化社区信息，格式为每一行对应一个节点与所属社区的对应。Map 函数将每个节点划分到自己节点编号的社区中，得到节点与所属社区的对应，Reduce 函数直接输出输入的键值对。

Map 函数的输入类型< IntWritable, IntWritable >以 TextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为节点 ID，value 为节点的邻接点信息。Map 函数将输入的节点 ID 划分到相同节点 ID 的社区 ID 中，输出类型为< IntPair,IntWritable>中的<key,value>，其中 key 为输入 key 节

点，value 为输入 key 节点的所属社区 ID。

Reduce 函数的输入类型为< IntPair,Iterable<IntWritable >>，输入<key,values>中 key 为节点。Reduce 函数直接输出输入的键值对，输出类型为< IntPair,IntWritable>中的<key,value>，其中 key 为节点 ID，value 为节点 key 所属的社区 ID，最后以 TextOutputFormat 类输出，得到输入网络的每个节点的所属社区信息。

2. 初始化紧密度矩阵

输入文本为所有节点与邻接点的对应，格式为每一行对应一个节点的邻接信息，由节点 ID 和邻接点 ID 组成，中间使用分隔符分隔。输出文本为输入网络的紧密度矩阵，格式为每一行对应一个节点与其他节点的紧密度值。Map 函数解析输入文本，得到节点与其邻接点的对应，根据紧密度值的计算公式，计算得到两个节点之间的紧密度值，Reduce 函数直接输出输入的键值对。

Map 函数的输入类型< IntWritable, IntWritable >以 InitcmInputFormat 类（自定义类）将输入文本分片得到，输入<key,value>中 key 为节点 ID，value 为节点 ID。Map 函数将读入节点之间的邻接信息，存入邻接数组，然后再利用函数 CM，根据公式（8-2）计算输入 key 节点和 value 节点之间的紧密度值，输出类型为< IntPair,IntWritable>中的<key,value>，其中 key 为输入 key 节点和 value 节点组成的紧密度矩阵中的位置信息（自定义类型 IntPair），value 为输入 key 节点和 value 节点之间的紧密度值，初始化紧密度矩阵的 Map 函数如下所示。

初始化紧密度矩阵的 Map 函数：

```
linjie=ReadLinjie()
location=new IntPair(key,value)
cmvalue=CM(linjie[key],linjie[value])
      context.write(location,cmvalue)
```

Reduce 函数的输入类型为< IntPair,Iterable<IntWritable >>，输入<key,values>中 key 为紧密度矩阵的位置。Reduce 函数直接输出输入的键值对，输出类型为< IntPair,IntWritable>中的<key,value>，其中 key 为紧密度矩阵的位置，value 为紧密度矩阵位置对应的紧密度值，最后以 MatrixOutputFormat 类（自定义输出类，以输出位置 key 的位置，输出矩阵）输出，得到输入网络的紧密度矩阵。

3. 一次随机游走过程

在并行化的一次随机游走过程中，可以并行计算所有节点的一次随机游走过程，经过一次 mapreduce job 后，得到所有节点一次随机游走过程后的所属社区情况，对于多次的随机游走过程，需要多次迭代 mapreduce job 得到最终的节点的社区划分结果。

输入文本为所有节点与邻接点的对应，格式为每一行对应一个节点的邻接信息，由节点 ID 和邻接点 ID 组成，中间使用分隔符分隔。输出文本为经过一次随机游走之后的节点与所属社团的对应，格式为每一行对应一个节点的所属社团信息，由节点 ID 和所属社区编号组成，中间使用分隔符分隔。Map 函数根据公式(8-3)计算得到输入 key 节点对输入 value 社区的 CA 值，Reduce

函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一个节点对所有其邻接社区的 CA 值，并计算随机移动概率，经过比较，使得节点向随机概率最大对应的社团移动，得到此节点与所属社团的对应，一次随机游走的 mapreduce 过程图如图 8.2 所示。

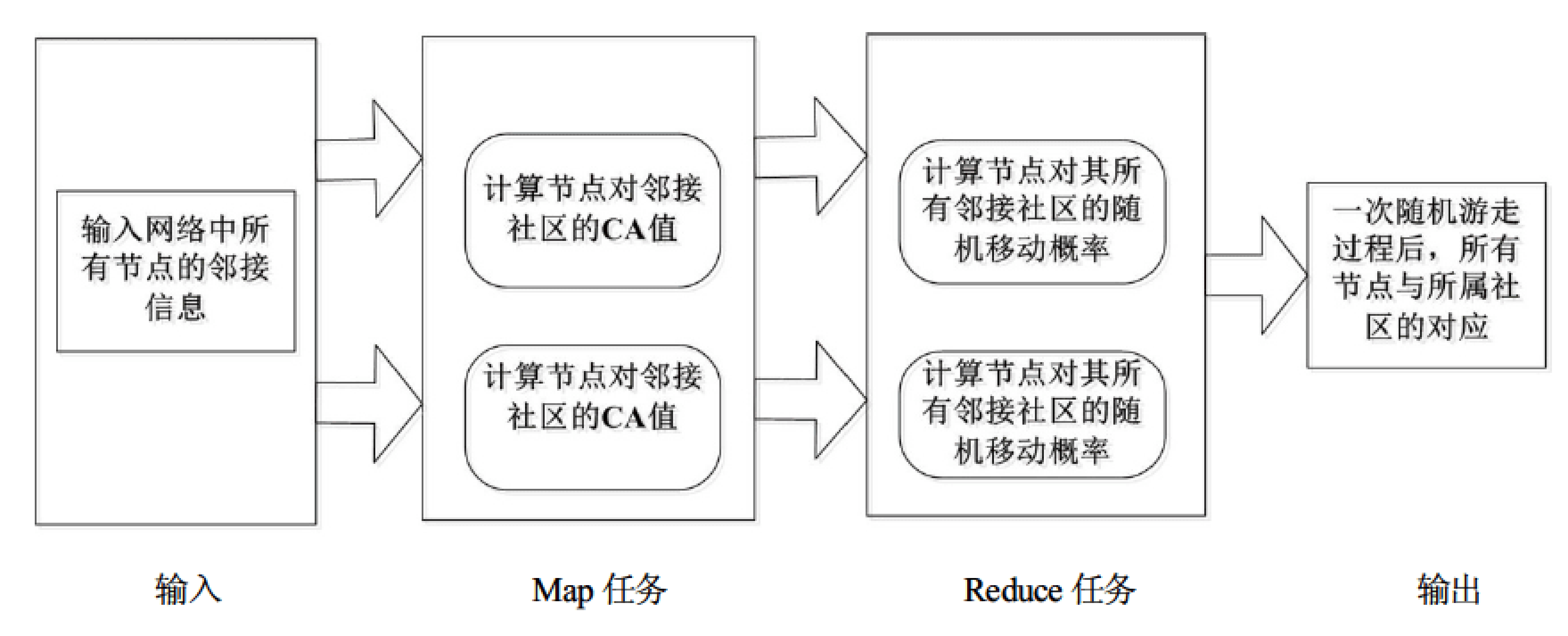


图 8.2　一次随机游走的 MapReduce 过程图

Map 函数的输入类型<IntWritable,IntWritable>以 RmsInputFormat 类（自定义类，使得 Map 的输入 key 为节点 ID，输入 value 为节点 ID 的邻接点的所属社区集合中的元素）将输入文本分片得到，输入<key,value>中 key 为节点 ID，value 为节点 key 的邻接社区。Map 函数读入节点之间的邻接信息和紧密度矩阵，函数 CA 根据公式（8-3）计算得到输入 key 节点对输入 value 社区的 CA 值，输出类型为< IntWritable,DoublePair >中的<key,value>，其中 key 为节点 ID，value 为输入 value 社区与计算得到的 CA 值组成的浮点数对，一次随机游走过程的 Map 函数如下所示。

一次随机游走过程的 Map 函数：

```
linjie=ReadLinjie()
CM=ReadCM()
cavalue=CA(key,value)
outputvalue=new DoublePair(value,cavalue)
context.write(key,outputvalue)
```

Reduce 函数的输入类型为< IntWritable,Iterable<DoublePair >>，输入<key,values>中 key 为节点 ID，经过 MapReduce 的 Shuffle 阶段，values 由同一节点 ID 对应的邻接社区与 CA 值组成的浮点数对组成。Reduce 函数根据公式（8-4）得到输入节点对邻接社区的随机移动概率，存入随机移动概率与邻接社区的映射，比较所有的随机移动概率，使得节点 ID 的所属社区为最大随机移动概率对应的社区，输出类型为< IntWritable,IntWritable>中的<key,value>，其中 key 为节点 ID，value 为节点 ID 所属的社区编号，最后以 TextOutputFormat 类输出，得到每个节点与所属社区的对应，一次随机游走过程的 Reduce 函数如图下所示。

一次随机游走过程的 Reduce 函数：

```
sum=0.0
 for(DoublePair val: values) do
       sum+=val.getright()
```

```
         end for
        ptoc=new HashMap<IntWritable,IntWritable>();
        for(DoublePair val: values) do
              p=val.getright()/sum
ptoc.put(p,val.getleft())
        end for
        outputvalue=pton.get(max(ptoc.keyset()))
        context.write(key,outputvalue)
```

8.4 AP 聚类算法的并行化实现

8.4.1 AP 聚类算法

仿射传播聚类算法是 J.Frey 在 2007 年发表在 Science 杂志上的一篇论文中提出的。这种算法的基本思想是通过数据点之间传递消息，自动发现聚类中心，并实现数据点的自动聚类。AP 聚类相比于经典的 K-means 聚类有一个明显的优点是，它不需要一开始假定聚类的类别是多少，也不需要去预设对聚类结果影响比较大的初始中心。AP 算法的聚类中心是在迭代中慢慢自动浮现出来的，同时 AP 聚类的聚类结果比 K-means 的要好。AP 聚类相比于谱聚类的一个优势是它在聚类速度上相对较快。一般谱聚类的算法时间复杂度是 $O(n^3)$，n 是数据点的个数。而 AP 聚类的时间复杂度是 $O(tn^2)$，其中 t 是算法迭代的次数。所以正常情况下，AP 聚类比谱聚类要快。AP 聚类是和其他算法（如 K-means）截然不同的一种算法，它并不在初始化阶段指定一些初始化中心，相反，它把每个数据点都同等地看作是潜在的聚类中心（在 AP 算法中，这些潜在的聚类中心称为 exemplars）。AP 聚类通过把数据点看做网络上的一个节点，并设置一些方法在节点之间传递实值的信息，直到一些足够好的聚类中心和整体上看来足够好的聚类结果浮现出来。

1. 计算相似度矩阵

AP 聚类需要的输入是一个由节点之间的相似度组成的一个矩阵 S，在这个矩阵中的每个元素 S(I,j)表示节点 i 与 j 之间的相似度，同时也表明了节点 j 作为 i 的聚类中心的合适程度。对于不同的数据集，相似值的计算方法可以根据数据集的特点来选择。以下是选用欧氏距离的负值作为相似值，将节点之间的邻接矩阵转换成节点之间的相似度矩阵，节点 i 和节点 j 之间的相似度利用公式（8-5）计算：

$$S(i,j) = -\sqrt{\sum_{k=1}^{n}(X_{ik} - X_{jk})^2} \qquad \text{式（8-5）}$$

如式（8-5）中 X_{ik} 表示节点之间的邻接矩阵 X 第 i 行第 k 列的元素，n 表示节点的总个数。

相比于要预设类簇的个数，仿射传播把一个对每个节点 k 赋一个实值 S(k,k)，即相似矩阵 S 对角线上第 k 行的元素。S(k,k)的值越大，就表明第 k 个节点作为聚类中心（exemplar）的可能性越大，这些对角线上的值称为“倾向值”（preference）。最终得到的聚类中心的个数一方面受到输入的倾向值的影响，另一个方面也会随着 AP 聚类的消息传递的过程而逐渐呈现。因为一开始 AP 聚类所有的节点作为聚类中心的可能性都是一样的，所以所有节点的“倾向值”都应该被赋予一个相同大小的值，这个值的设定会影响到最后聚类类簇（clusters）的个数。一般可以把这个值设为相似值的平均值（这样产生的类簇的数目处于一个中等大小的级别），或者相似值的最小值（这样产生的类簇的数目处于一个较少的级别）。所以如果你想要你的类簇的数目比较小，就可以把这个“倾向值”设成一个比较小的值。

2. 执行 AP 聚类算法

AP 聚类算法定义节点之间传递两类信息，并且这两类信息会被引入用于两套不同的竞争机制中。这两类信息分别称为吸引度（responsibility）和归属度（availability）。

吸引度是从节点 i 传递到候选作为其聚类中心的节点 k 的信息，称为节点 k 对于节点 i 的吸引度，其值记为 r(i,k)。吸引度 r(i,k)反映的是节点 k 通过与其他节点 k' 竞争，作为适合节点 i 的聚类中心的程度。r(i,k)的计算需要引入节点 i 对于其他潜在的候选聚类中心节点 k' 的归属度 $a(i,k')$ 来触发竞争，r(i,k)的计算如式（8-6）所示：

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\} \qquad \text{式（8-6）}$$

归属度是从候选的类簇中心节点 k 传递到节点 i 的信息，称为节点 i 对于节点 k 的归属度，其值记为 a(i,k)。归属度 a(i,k)反映的是节点 i 选择节点 k 作为其类簇中心的适合程度，这种竞争的思想是：如果节点 k 作为其他节点 i' 的类簇中心的合适程度很大，则节点 k 作为节点 i 的类簇中心的合适程度也会较大。a(i,k)计算的是引入是将候选类簇中心的节点 k 对其他节点 i' 的吸引度作为比较，计算 a(i，k)如式（8-7）所示：

$$a(i,k) \leftarrow \min\{0, r(k,k) + \sum_{i' \neq \{i,k\}} \max\{0, r(i',k)\}\} \qquad \text{式（8-7）}$$

自发的归属度 a(k,k)被更新的方法不同，如式（8-8）所示：

$$a(k,k) \leftarrow \sum_{i' \neq \{i,k\}} \max\{0, r(i',k)\} \qquad \text{式（8-8）}$$

这个消息反映的是节点 k 作为类簇中心的能力，主要是基于非负的其他节点发给候选节点 k 的吸引度。

整个 AP 聚类的算法流程如下：

（1）初始化，将所有的 availability 的值 a(i,k)全部赋值为零，输入相似矩阵 s，其中 s(i,k)是节点 i 与节点 k 之间的相似值，而 s(k,k)则是 k 点作为候选类簇中心的倾向值。

（2）计算节点 k 对于节点 i 的吸引度，其计算公式为：

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}$$

（3）计算节点 i 对于节点 k 的归属度，其计算公式为：

$$a(i,k) \leftarrow \min\{0, r(k,k) + \sum_{i' \neq \{i,k\}} \max\{0, r(i',k)\}\}$$

$$a(k,k) \leftarrow \sum_{i' \neq \{i,k\}} \max\{0, r(i',k)\}$$

（4）直到类簇中心不再发生变化，或者已经完成了指定的迭代次数后，停止计算，否则重复第（2）步和第（3）步。

（5）对角线上的值 a(k，k)+r(k，k)>0 的节点 k 为自动发现的类簇中心，而对于节点 i，使 a(i,k)+r(i,k)最大的候选类簇中心为其真正归属的类簇中心。

8.4.2 AP 聚类算法在 MapReduce 上的实现

1. 计算相似度矩阵

输入文本为节点之间的邻接矩阵，格式为每一行对应一个节点的邻接节点信息。输出文本为节点之间的相似度矩阵，格式为每一行对应一个节点与所有节点之间的相似值。Map 函数得到公式（8-5）中 $(X_{ik}-X_{jk})^2$ 的结果，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并计算值相加、开平方和求负得到最终的相似值。

Map 函数的输入类型<IntPair,FolatPair>（IntPair 类中有两个整数属性，FloatPair 类中有两个浮点数属性）以 MatrixInputFormat 类（将输入矩阵以一行和一列为一个分片，每一个分片中行号和列号组成 key，行列对应位置的两个矩阵值组成 value）将输入文本分片得到，输入<key,value>中 key 为输入文本中矩阵的行列号，value 为行列对应位置的两个矩阵值。Map 函数中 value 的 getLeft()和 getRight()方法获得 value 中两个浮点数，sub 函数为两个参数相减后平方的结果，输出类型为< IntPair,FloatWritable >中的<key,value>，其中 key 为相似度矩阵的行列号，value 为函数 sub 计算后得到的值，实现框架如下所示。

计算节点之间的相似度矩阵的 Map 函数：

```
dif=sub(value.getLeft(),value.getRight())
context.write(key,dif)
```

Reduce 函数的输入类型为< IntPair,Iterable<FloatWritable >>，输入<key,values>中 key 为相似度矩阵的行列号，经过 MapReduce 的 Shuffle 阶段 values 由多个位置对应的两个矩阵值 $(X_{ik}-X_{jk})^2$ 计算得到的值组成。Reduce 函数相加所有 values 中的值后开平方和加负号，得到公式（8-5）计算后的相似值，输出类型为< IntPair,IntWritable>中的<key,value>，其中 key 为相似度矩阵的行列号，value 为相似度矩阵位置 key 的值，最后以 MatrixOutputFormat 类输出，得到节点之间的相似度矩阵，实现框架如下所示。

计算节点之间的相似度矩阵的 Reduce 函数：

```
        sum=0.0
        for(FloatWritable val: values) do
              sum+=val.get()
        end for
        result=-(Math.sqrt(sum))
        context.write(key,result)
```

2. 执行 AP 聚类算法

AP 聚类的整个过程为多次迭代计算矩阵值，直到达到迭代次数或聚类中心连续多次不再发生改变，需要进行多次迭代的 mapreduce job，其中一次迭代包括计算吸引度矩阵、计算归属度矩阵和转换归属度矩阵的列文件到归属度矩阵的行文件，当达到迭代次数或聚类中心连续多次不再发生改变后，发现聚类中心并划分节点。

（1）计算吸引度矩阵

输入文本为节点之间的相似度矩阵和初始化或上次迭代产生的归属度矩阵，相似度矩阵格式为每一行对应一个节点与所有节点的相似度信息，每一行按顺序由矩阵行号、第一列相似度值和相似度矩阵标志字符 s 组成的字符串、第二列相似度值……最后一列相似度值组成，行号和相似值之间用“\t”字符隔开，相似值之间用空格字符隔开，归属度矩阵的格式与相似度矩阵的格式类似，只是每一行第一列是第一列相似度值和归属度矩阵标志字符 a 组成的字符串。输出文本为节点之间的吸引度矩阵，格式为每一行对应一个节点与所有节点之间的归属度信息，每一行按顺序由矩阵行号、第一列归属度值……最后一列归属度值，分隔符的格式与相似度矩阵一样。

分析公式（8-6），可以看出行号 i 是固定的，而列号 k' 是变化的，所以可以把吸引度矩阵中的某一行 i 的所有值放在一起计算，即 Map 函数将输入文本中的相似度矩阵和归属度矩阵的每一行内容转换成公式（8-6）所需要的格式，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一行号的相似度值和归属度值，根据公式（8-6）计算得到同一行号的吸引度值，计算吸引度矩阵的 mapreduce 过程图如图 8.3 所示。

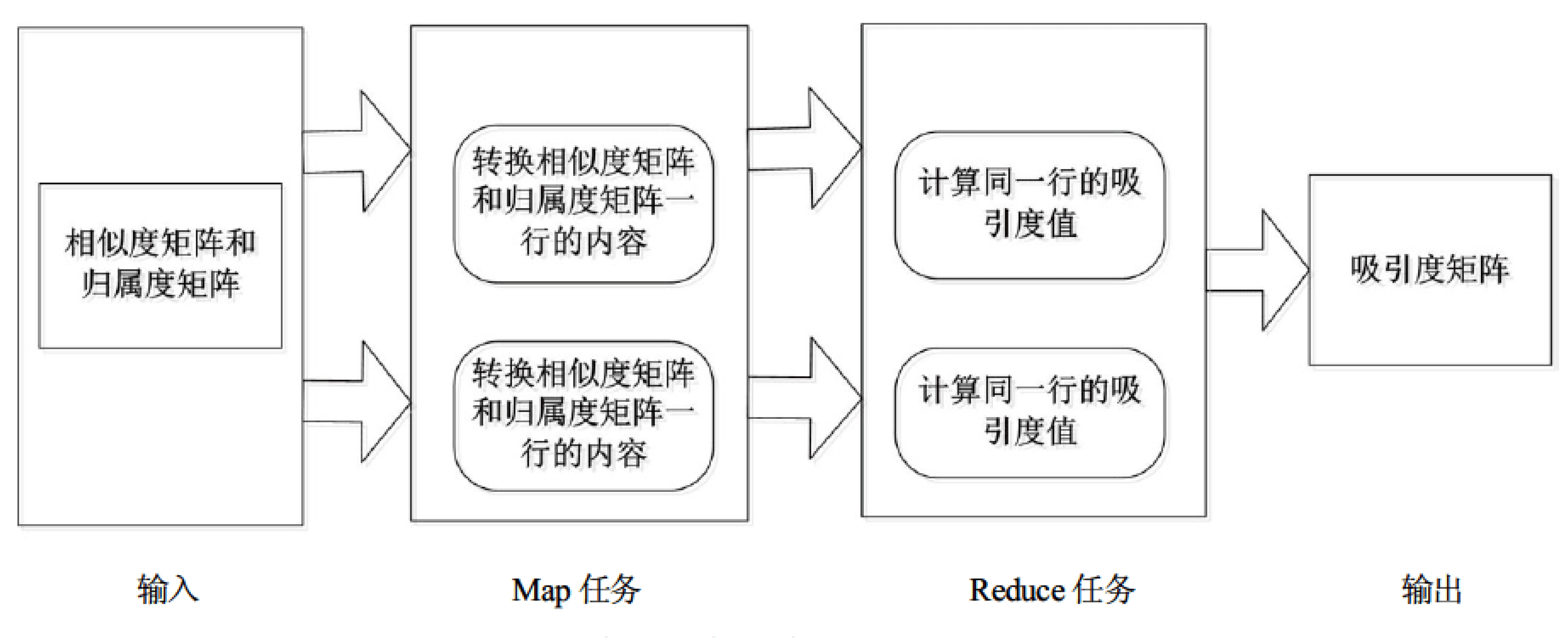


图 8.3　计算吸引度矩阵的 MapReduce 过程图

Map 函数的输入类型<Text,Text>以 KeyValueTextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为输入文本相似度矩阵或归属度矩阵的行号，value 为输入文本相似度矩阵或归属度矩阵中 key 行所有值构成的文本串。Map 函数中将 Text 类型输入 key 转换成 IntWritable 类型

的输出 key，输出类型为< IntWritable,Text >中的<key,value>，其中 key 为矩阵的行号，value 为矩阵中 key 行所有值构成的文本串，实现框架如下所示。

计算吸引度矩阵的 Map 函数：

```
row=new IntWritable()
row.set(Integer.parseInt(key.toString()))
context.write(row,value)
```

Reduce 函数的输入类型为< IntWritable,Iterable<Text >>，输入<key,values>中 key 为矩阵的行号，经过 MapReduce 的 Shuffle 阶段 values 由 key 行号的相似度矩阵行的所有值构成的文本串和归属度矩阵行的所有值构成的文本串组成。Reduce 函数中将输入的行文本串加入到相应矩阵的列表中，经过函数 Rcompute 计算[根据公式（8-6）计算]，得到新产生的吸引度矩阵中 key 行的所有吸引度值，输出类型为< IntWritable,Text>中的<key,value>，其中 key 为吸引度矩阵的行号，value 为吸引度 key 行中所有吸引度值构成的文本串，最后以 MultipleTextOutputFormat 类输出，得到节点之间的吸引度矩阵，实现框架如下所示。

计算吸引度矩阵的 Reduce 函数：

```
List S=empty,A=empty
while(values.hasNext()) do
     rowc=values.next().toString().split(“ ”)
     (S,A)=addSA(rowc)
end while
new_rowc=Rcompute(S,A)
context.write(key,new_rowc)
```

（2）计算归属度矩阵

输入文本为本次迭代产生的节点之间的吸引度矩阵，吸引度矩阵格式同上。输出文本为节点之间的归属度矩阵，归属度矩阵格式和上节的归属度矩阵类似，不同是输出文本的每一行表示归属度矩阵的一列内容。

分析公式（8-7）和公式（8-8），可以看出行号 i' 是变化的，而列号 k 是固定的，所以我们可以把归属度矩阵中的某一列 k 的所有值放在一起计算，即 Map 函数将输入文本中的吸引度矩阵的每一行的全部吸引度值分开，得到列号与吸引度值的对应，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一列的吸引度值根据公式（8-7）和公式（8-8）得到同一列的归属度值。

Map 函数的输入类型<Text,Text>以 KeyValueTextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为输入文本吸引度矩阵的行号，value 为输入文本吸引度矩阵中 key 行所有值构成的文本串。Map 函数循环遍历得到一行的每一个值，并将列号与吸引度值的对应输出，输出类型为< IntWritable,Text >中的<key,value>，其中 key 为矩阵的列号，value 为矩阵中输入 key 行输出 key 列的吸引度值与行号组成的文本串，实现框架如下所示。

计算归属度矩阵的 Map 函数：

```
colc=value.toString().split(“ ”)
coln=new IntWritable()
for(num in colc.length) do
```

```
            coln.set(num)
            colv=new Text(colc[num]+key)
            context.write(coln,colv)
        end for
```

Reduce 函数的输入类型为< IntWritable,Iterable<Text >>，输入<key,values>中 key 为矩阵的列号，经过 MapReduce 的 Shuffle 阶段 values 由 key 列号的所有吸引度值和自己行号构成的文本串组成。Reduce 函数中将输入的列文本串加入到吸引度矩阵列表中，经过函数 Acompute 计算[根据公式（8-7）和公式（8-8）计算]，得到新产生的归属度矩阵中 key 列的所有归属度值，输出类型为< IntWritable,Text>中的<key,value>，其中 key 为归属度矩阵的列号，value 为归属度矩阵 key 列中所有归属度值构成的文本串，最后以 MultipleTextOutputFormat 类输出，得到节点之间的归属度矩阵，实现框架如下所示。

计算归属度矩阵的 Reduce 函数：

```
    List R=empty
    while(values.hasNext()) do
        R.add(values.next().toString())
    end while
    new_colc=Acompute(R)
    context.write(key,new_colc)
```

（3）转换归属度矩阵的列文件到归属度矩阵的行文件

输入文本为上节产生的节点之间的归属度矩阵，输出文本为节点之间的归属度矩阵行文件，归属度矩阵格式和计算吸引度矩阵的输入归属度矩阵格式相同。Map 函数将输入文本中归属度矩阵的每一列的全部归属度值分开，得到行号与归属度值的对应，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一行的归属度值组合输出，转换归属度矩阵的列文件到归属度矩阵的行文件的 mapreduce 过程图如图 8.4 所示。

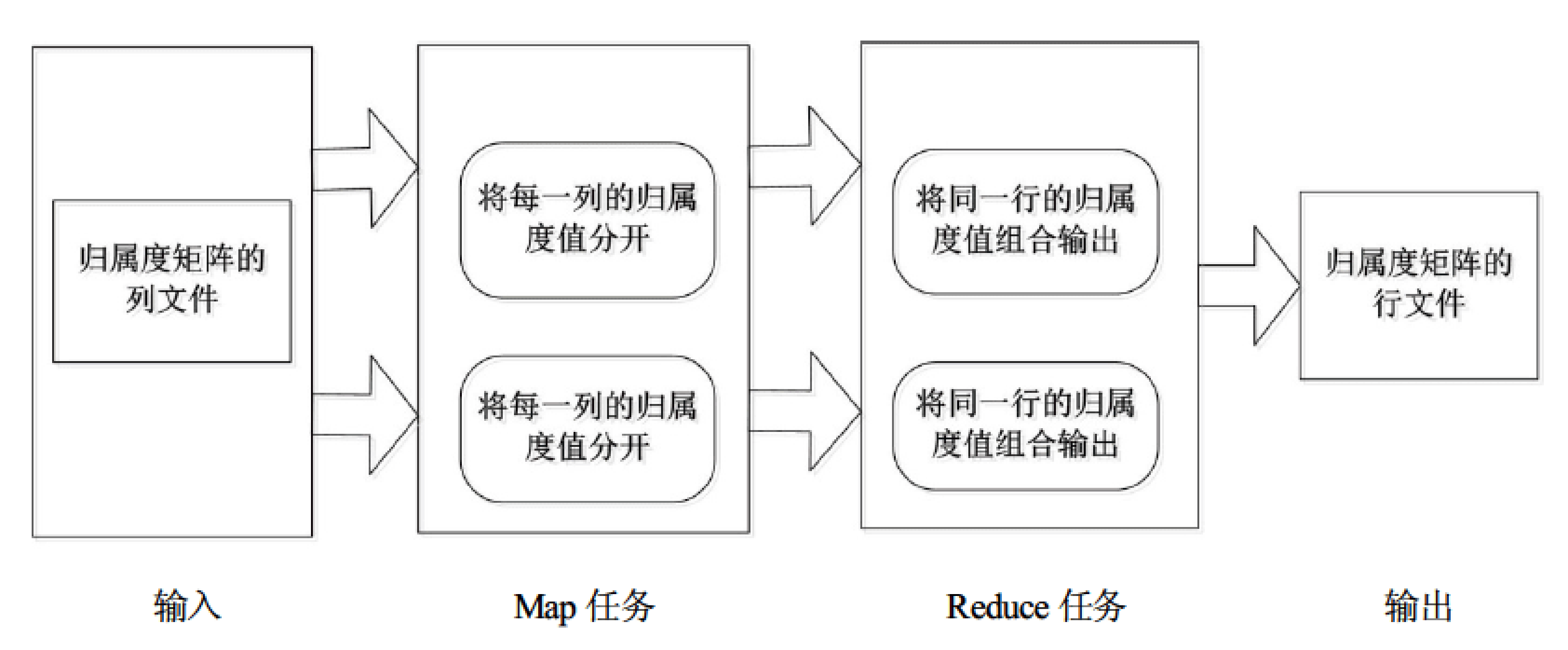


图 8.4　转换归属度矩阵的列文件到归属度矩阵的行文件的 MapReduce 过程图

Map 函数的输入类型<Text,Text>以 KeyValueTextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为输入文本归属度矩阵的列号，value 为输入文本归属度矩阵中 key 列所有值构成的文本串。Map 函数循环遍历得到一列的每一个值，并将行号与吸引度值的对应输出，输出类型为< IntWritable,Text >中的<key,value>，其中 key 为矩阵的行号，value 为矩阵中输入 key 列输出 key 行的归属度值和列号构成的文本串，实现框架如下所示。

转换列文件到行文件的 Map 函数：

```
colc=value.toString().split(" ")
rown=new IntWritable()
for(num in colc.length) do
    rown.set(num)
    rowv=new Text(colc[num]+key)
    context.write(rown,rowv)
end for
```

Reduce 函数的输入类型为< IntWritable,Iterable<Text >>，输入<key,values>中 key 为矩阵的行号，经过 MapReduce 的 Shuffle 阶段 values 由 key 行号的所有归属度值和自己列号构成的文本串组成。Reduce 函数中 trans 函数将输入的 key 行的所有归属度值组合成 Text 类的文本串，得到归属度矩阵中 key 行的所有归属度值，输出类型为< IntWritable,Text>中的<key,value>，其中 key 为归属度矩阵的行号，value 为归属度矩阵 key 行中所有归属度值构成的文本串，最后以 MultipleTextOutputFormat 类输出，得到节点之间的归属度矩阵，实现框架如下所示。

转换列文件到行文件的 Reduce 函数：

```
List R=empty
while(values.hasNext()) do
      R.add(values.next().toString())
end while
new_rowc=trans(R)
context.write(key,new_rowc)
```

（4）发现聚类中心

输入文本为最终产生的节点之间的吸引度矩阵和归属度矩阵，输出文本为发现的聚类中心，格式为每一行对应一个发现的聚类中心信息，由聚类中心 ID 和聚类中心行在吸引度矩阵的对角线值与归属度矩阵的对角线值相加和组成，中间使用分隔符分隔。Map 函数解析输入矩阵的一行值，得到每一行的对角线值，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一行吸引度矩阵和归属度矩阵的对角线值相加，判断是否为聚类中心。

Map 函数的输入类型<Text,Text>以 KeyValueTextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为输入文本归属度矩阵或吸引度矩阵的行号，value 为输入文本归属度矩阵或吸引度矩阵中 key 行所有值构成的文本串。Map 函数解析输入矩阵的一行值，得到每一行的对角线值，输出类型为< IntWritable,Text >中的<key,value>，其中 key 为矩阵的行号，value 为矩阵中输入 key 行的对角线值。

Reduce 函数的输入类型为< IntWritable,Iterable<Text >>，输入<key,values>中 key 为矩阵的行号，经过 MapReduce 的 Shuffle 阶段 values 由 key 行的吸引度矩阵对角线值和归属度矩阵对角线值组成。Reduce 函数中将两个对角线值解析出来相加，和大于 0 则节点 key 为发现的聚类中心，输出类型为< IntWritable,Text>中的<key,value>，其中 key 为发现的聚类中心节点 ID（矩阵的行号），value 为发现的聚类中心吸引度矩阵对角线值和归属度矩阵对角线值相加的和，最后以 MultipleTextOutputFormat 类输出，得到发现的聚类中心。

（5）划分节点

输入文本为最终产生的节点之间的吸引度矩阵和归属度矩阵，输出文本为所有节点与所属聚类中心的对应，格式为每一行对应一个节点所属的聚类中心信息，由节点 ID 和聚类中心 ID 组成，中间使用分隔符分隔。Map 函数解析输入矩阵的一行值，得到每一行的中所有聚类中心列的值，Reduce 函数将经过 MapReduce 的 Shuffle 阶段归并得到的同一行吸引度矩阵和归属度矩阵的所有聚类中心列的值相加，节点所属于相加和最大的聚类中心。

Map 函数的输入类型<Text,Text>以 KeyValueTextInputFormat 类将输入文本分片得到，输入<key,value>中 key 为输入文本归属度矩阵或吸引度矩阵的行号，value 为输入文本归属度矩阵或吸引度矩阵中 key 行所有值构成的文本串。Map 函数中函数 Readcenters 读取发现的聚类中心文件，得到聚类中心列表，遍历 key 行的所有值，函数 join 将 key 行中所有聚类中心列的值组合成文本串输出，输出类型为< IntWritable,Text >中的<key,value>，其中 key 为矩阵的行号，value 为矩阵中输入 key 行中所有聚类中心列的值组成的文本串，实现框架如下所示。

划分节点的 Map 函数：

```
List centers=Readcenters()
colcenter=new Text()
rowc=value.toString().split(" ")
for(num in rowc.length) do
   if (centers.contains(num))
      colcenter=join(colcenter,rowc[num])
   end if
end for
context.write(key,colcenter)
```

Reduce 函数的输入类型为< IntWritable,Iterable<Text >>，输入<key,values>中 key 为矩阵的行号，经过 MapReduce 的 Shuffle 阶段 values 由 key 行的吸引度矩阵所有聚类中心列的值和归属度矩阵所有聚类中心列的值组成。Reduce 函数中将 key 行中吸引度矩阵和归属度矩阵相同列的两个对应值解析出来并相加，找出使吸引度值和归属度值相加和最大的聚类中心列，则第 key 个节点属于此聚类中心，输出类型为< IntWritable,Text>中的<key,value>，其中 key 为节点 ID（矩阵的行号），value 为节点 key 所属的聚类中心，最后以 MultipleTextOutputFormat 类输出，得到所有节点与所属聚类中心的对应。

8.5 实验与分析

8.5.1 实验环境

本实验用到的 Hadoop 集群环境包含 3 台机器，是一个典型的主从式（Master-Slaves）结构。集群包含一个主控节点（Master）和两个从属节点（Slave）。在主从结构中，主节点一般负责集群

管理、任务调度和负载平衡等，而从节点则执行来自主节点的计算和存储任务。集群环境的具体软硬件和网络配置情况如表 8.2 所示。主控节点的硬件配置最高，其余两台从属节点硬件配置相同，搭载的操作系统都是 Ubuntu11.10。Hadoop 平台基于 jdk1.6.0_02 搭建，Hadoop 使用的是目前的稳定版本，版本号是 0.20.203.0。

表 8.2　集群环境配置表

序号	主机名	网络地址	硬件参数	操作系统
A	jobtracker	192.168.1.2	CPU：Intel Xeon W3505 内存：2GB DDR3*3 硬盘：SATA 1TB	Ubuntu11.10
B	tasktracker1	192.168.1.7	CPU：Intel Core2 E7500 内存：2GB DDR3 硬盘：SATA 320GB	Ubuntu11.10
C	tasktracker2	192.168.1.12	CPU：Intel Core2 E7500 内存：2GB DDR3 硬盘：SATA 320GB	Ubuntu11.10

8.5.2　实验与结果分析

本节按照规模将测试集分为较小规模和较大规模两类进行实验，较小规模测试集包括 karate 网络、dolphin 网络和 football 网络，较大规模测试集包括 CA-HepPH 和 Enron 邮件通信网络。

1. 较小规模测试集实验

实验采用 EQ 函数来评价重叠社团结构，它是 shen 等人为了能在树状图中选择某一层作为社团发现结果而提出来的 Q 函数的扩展，如式（8-9）所示：

$$EQ=\frac{1}{2m}\sum_{i}\sum_{v\in C_i,u\in C_i}\frac{1}{O_vO_u}(A_{vu}-\frac{k_vk_u}{2m})\qquad \text{式（8-9）}$$

其中 m 是边的数目，i 表示社团编号，O_v 和 O_u 分别表示节点 v 和 u 所属社团的数量，A 是网络的邻接矩阵。无向图中，k_v 和 k_u 分别表示节点 v 和 u 的度。当社团结构是非重叠时，EQ 函数的值和 Q 函数值一致。Shen 等人还在论文中指出，较高的 EQ 代表了较强的重叠社团结构。下面围绕 karate 网络、dolphin 网络和 football 网络这三个真实网络数据来进行测试。

karate 网络是一个俱乐部网络，具有 34 个节点和 78 条边。该俱乐部因为教练（节点 1）和主管（节点 34）之间发生争执而分裂成两个小俱乐部。该网络由于社团结构已知，曾被广泛用作检验社团发现算法的数据集。一般的社团发现算法将该网络分为 2~4 个社团，节点 3、9、10、24 常被作为重叠节点。表 8.3 中列出了并行化 RMS 算法、并行化 AP 聚类算法和其他两种算法 CPM、

COPRA 对 karate 网络的实验结果对比，从运行时间可以看到并行化 RMS 算法和并行化 AP 算法均比 CPM 算法耗时长，但是低于 COPRA 算法，从社团结构的 EQ 值可以看到并行化 RMS 算法划分的社团结构是最好的，高于其他三种算法，并行化 AP 算法由于是非重叠的社团发现算法，社团划分效果处于 CPM 算法和 COPRA 算法之间。

表 8.3　4 种算法作用于 karate 网络的 EQ 和运行时间

	并行化 RMS	并行化 AP	CPM	COPRA
EQ	0.365	0.158	0.115	0.188
运行时间（单位为 ms）	56	34	8	168

dolphin 网络是新西兰某海湾的 62 头海豚之间联系关系而形成的网络，具有 62 个节点和 78 条边，也是一个被广泛使用的社会网络数据集。常见算法一般将其分作 4 个社团。表 8.4 中列出了并行化 RMS 算法、并行化 AP 聚类算法和其他两种算法 CPM、COPRA 对 dolphin 网络的实验结果对比，从运行时间可以看到并行化 RMS 算法和并行化 AP 算法均比 CPM 算法耗时长，但是低于 COPRA 算法，，从社团结构的 EQ 值可以看到并行化 RMS 算法划分的社团结构是最好的，高于其他三种算法，并行化 AP 算法划分的社团结构较之其他三种算法最差。

表 8.4　4 种算法作用于 dolphin 网络的 EQ 和运行时间

	并行化 RMS	并行化 AP	CPM	COPRA
EQ	0.457	0.276	0.288	0.373
运行时间（单位为 ms）	118	96	11	228

football 是 2000 年秋季美国高校橄榄球甲级常规赛季的关系数据，节点代表球队，边表示两个球队参加过同一场比赛。这些球队可以分成 8~12 个联盟，联盟内部比赛的频率高于联盟间的比赛频率，因此也存在社区结构。表 8.5 中列出了并行化 RMS 算法、并行化 AP 聚类算法和其他两种算法 CPM、COPRA 对 football 网络的实验结果对比，从运行时间可以看到并行化 RMS 算法和并行化 AP 算法均比 CPM 算法耗时长，但是低于 COPRA 算法，，从社团结构的 EQ 值可以看到并行化 RMS 算法划分的社团结构是最好的，高于其他三种算法，并行化 AP 算法相对于其他三种算法较差。

表 8.5　4 种算法作用于 football 网络的 EQ 和运行时间

	并行化 RMS	并行化 AP	CPM	COPRA
EQ	0.563	0.538	0.559	0.550
运行时间（单位为 ms）	237	162	44	279

2. 较大规模测试集实验

CA-HepPH 是一个科学家合作网络，具有 12 008 个节点和 237 010 条边。表 8.6 中列出了并行化 RMS 算法、并行化 AP 聚类算法和其他两种算法 CPM、COPRA 对 CA-HepPH 网络的实验结果对比，从运行时间可以看到 CPM 算法已经无法运行，并行化 RMS 算法和并行化 AP 算法均比 COPRA 算法算法耗时长，从社团结构的 EQ 值可以看到并行化 RMS 算法划分的社团结构是最好的，高于其他三种算法，并行化 AP 算法划分的社团结构次之。

表 8.6　4 种算法作用于 CA-HepPH 网络的 EQ 和运行时间

	并行化 RMS	并行化 AP	CPM	COPRA
EQ	0.278	0.196	-	0.171
运行时间（单位为 s）	76	89	∞	30.40

Enron 是一个邮件通信网络，节点是邮箱，若两个邮箱之间至少有一次邮件往来则这两个节点之间存在边。该网络具有 36 692 个节点和 367 662 条边。表 8.7 中列出了并行化 RMS 算法、并行化 AP 聚类算法和其他两种算法 CPM、COPRA 对 Enron 网络的实验结果对比，从运行时间可以看到 CPM 算法已经无法运行，并行化 RMS 算法和并行化 AP 算法均比 COPRA 算法耗时长，从社团结构的 EQ 值可以看到并行化 RMS 算法划分的社团结构是最好的，高于其他三种算法，并行化 AP 算法划分的社团结构次之。

表 8.7　4 种算法作用于 Enron 网络的 EQ 和运行时间

	并行化 RMS	并行化 AP	CPM	COPRA
EQ	0.219	0.158	-	0.130
运行时间单位(s)	96	128	∞	50.99

特别地，当处理邮件通信网络时，CPM 算法已无法运行，而并行化的 RMS 算法和并行化 AP 算法可以在有限时间内完成，所以并行化的 RMS 算法和并行化的 AP 算法能够在有限的时间内处理大规模数据。当然，由于实际情况下，集群会受到机器本身的状态、文件系统的 I/O 读写以及网络通信时延等情况影响，它的运行时间要比预期的长。

8.6 本章小结

自 2002 年 Girvan 和 Newman 提出 GN 算法以来，社区发现发展迅速，从非重叠社区结构到重叠社区、层次社区结构，从起初的模块化函数到 EQ 函数，从起初使用不符合真实网络特征的测试网络到使用尽可能符合真实网络特征的 LFR 人工网络，以及派系过滤算法、LFM 算法、GCE

算法等一系列社区算法的不断提出，都直接推动了社区发现成为近年复杂网络研究领域的热门问题。虽然社区发现算法的时间复杂度不断降低，效果也越来越好，但许多算法都有其局限性，复杂网络中的社区发现问题迄今并未被完美解决。

Google 提出的 Map/Reduce 概念是当前比较流行的分布式计算框架。本章研究两种在 Map/Reduce 上实现的算法：并行化 RMS 算法和并行化 AP 聚类。分别实现这两种算法在 3 台机器组成的 Hadoop 集群上的分布式计算。

通过学习研究 RMS 算法和 AP 算法，实现了 RMS 算法和 AP 算法的 MapReduce 版本，使得算法能够有效解决大规模复杂网络的社区发现问题；通过在两种算法上测试不同的测试集，对比并行化 RMS 算法、并行化 AP 算法、CPM 算法和 COPRA 算法的社区划分质量和运行时间，发现并行化 RMS 算法和并行化的 AP 算法能够有效地处理大规模网络的社团发现问题。

第 9 章 数据统一访问与转换平台

在信息技术飞速发展的今天，企业的信息化发展过程中经常会生成 TB 级别的数据，随着时间的累积和业务的扩展，数据来源涵盖了互联网中可以捕获的各种类型数据，网站、社交媒体、交易型商业数据以及其他商业环境中创建的数据。这些数据源大多存在于不同的硬件和软件环境中，其数据以各种格式来表现。由于数据源的差异以及数据量的庞大，对这些数据的统一处理和分析成为了目前数据应用中面对的首要问题。

9.1 应用背景介绍

随着 20 世纪信息系统集成的兴起，数据集成或数据的统一访问访问作为一种资源整合的理念和方式逐步受到重视。企业在信息化的建设过程中，建立了由不同核心技术构建的信息系统以及相应的数据库，由此构成了一个个的异构数据源。如何通过一个集成系统，将企业内部和外部的异构数据源进行整合，提高资源的利用效率，为企业和事业单位提供有效的数据支持，是现代信息系统建设面临的巨大挑战。

目前实现数据共享有两种途径，一是数据转换，另一种是数据集成。第一种途径是物理意义上的数据集中，但是一方面它需要在硬件及相关软件上进行巨大的投资，另一方面进行海量数据迁移和管理也存在相当大的风险，相应访问速度也可能不理想。第二种途径属于逻辑集中。这种途径能够充分利用现有系统，对信息资源进行分布存储、分散管理、统一访问接口，更能适应信息系统的发展现状。两种方式各有优缺点。

经历了二十多年的信息发展，关于信息数据的统一访问已经有了诸多的理论和技术实现，研究者提出了许多数据统一访问的体系结构和实现方案。从模型上分有三种，分别是联邦方式、数据仓库和中间件方式。从集成技术上分，则异构数据库集成技术主要有数据的迁移和转换、多数据库系统和使用中间件。

但是目前通用数据统一访问访问和转换平台的研究尚处于起步阶段，国外一些著名的数据库公司开发了相应的中间件产品，用于解决异构数据集成问题。但使用这些中间件产品需要做大量的数据接口开发工作，并且现阶段国内还缺乏比较完整的数据整合产品。

同时现有的数据编程技术通常是或多或少地针对特定数据源类型而设计的，但是现实世界的应用中，数据往往都来自于多种数据源。可以忽略数据来源普通数据的表达集则能为应用开发者提供一种简单、统一的编程模型。

数据的统一访问访问与转换技术的提出正是为了实现网络信息资源的共享与统一访问。数据的统一访问与转换平台收集、组织并集成来自不同数据源的数据，并为应用程序员提供统一、规范的数据访问形式，实现对各类分布的异构数据源进行透明访问。数据的统一访问访问与转换平台的目标是对异构数据源的统一访问和应用，即将访问请求分解到各个不同的数据源中，再将返回的异构结果进行统一整合转换，给应用程序的设计者提供了一个统一的数据源访问接口，并为后续的数据分析奠定基础。

服务数据对象（Service Data Objects，SDO）技术正逐步成为数据集成研究的热点。SDO 与语言无关，可在一系列编程语言中使用，目标是创建一个数据访问层实现异构数据的统一访问。通过使用 SDO 技术，应用程序编程人员可以采用简单易用的统一方式访问和操作来自异构数据源的数据，并使用跨平台标记语言 XML 来描述信息资源模型。服务数据对象的出现使得对各种不规则的数据信息（当然也包括规则信息）集成为一个可以忽略数据来源的普通数据的表达集成为可能。

随着 SOA 理念的流行和 Web Service 等技术的广泛应用，我们发现在越来越多的系统中，需要访问各种不同的底层数据，这些数据包括关系型数据库、EJB 组件、XML 文件或数据库、Web 服务、JSP 页面数据等。为了能够访问和操作这些数据，开发人员必须了解针对不同数据源操作的规范和 API。服务数据对象技术为我们提供了统一的数据访问应用开发框架，它提供了对多种企业信息系统（EIS）的统一数据访问，包括数据库、遗留应用程序（使用 JCA）、XML 或者是 Web 服务数据源。通过使用服务数据对象的一种独特而简单的数据模型，应用程序能够摆脱使用多种 API 和框架进行数据访问的复杂工作，使开发人员只需了解服务数据对象技术 API 即可操作各种异构数据源数据信息。

服务数据对象 SDO，是 BEA 和 IBM 共同发布的一项规范。SDO 是 Java 平台的一种数据编程架构和 API，它统一了不同数据源类型的数据编程，提供了对通用应用程序模式的健壮支持，并使应用程序、工具和框架更容易查询、读取、更新和检查数据。这里需要说明，SDO 不是一种针对数据访问和持久化的技术，而是一种数据编程架构和一组 API。SDO 主要用于简化数据编程，让开发人员能集中解决业务逻辑问题而不是底层技术。

服务数据对象是信息的容器，设计用于提升开放标准和互操作性。SDO 提供了在整个企业应用程序中表示信息的方法，包括表示层、业务逻辑层和持久层之间的通信，如图 9.1 所示。

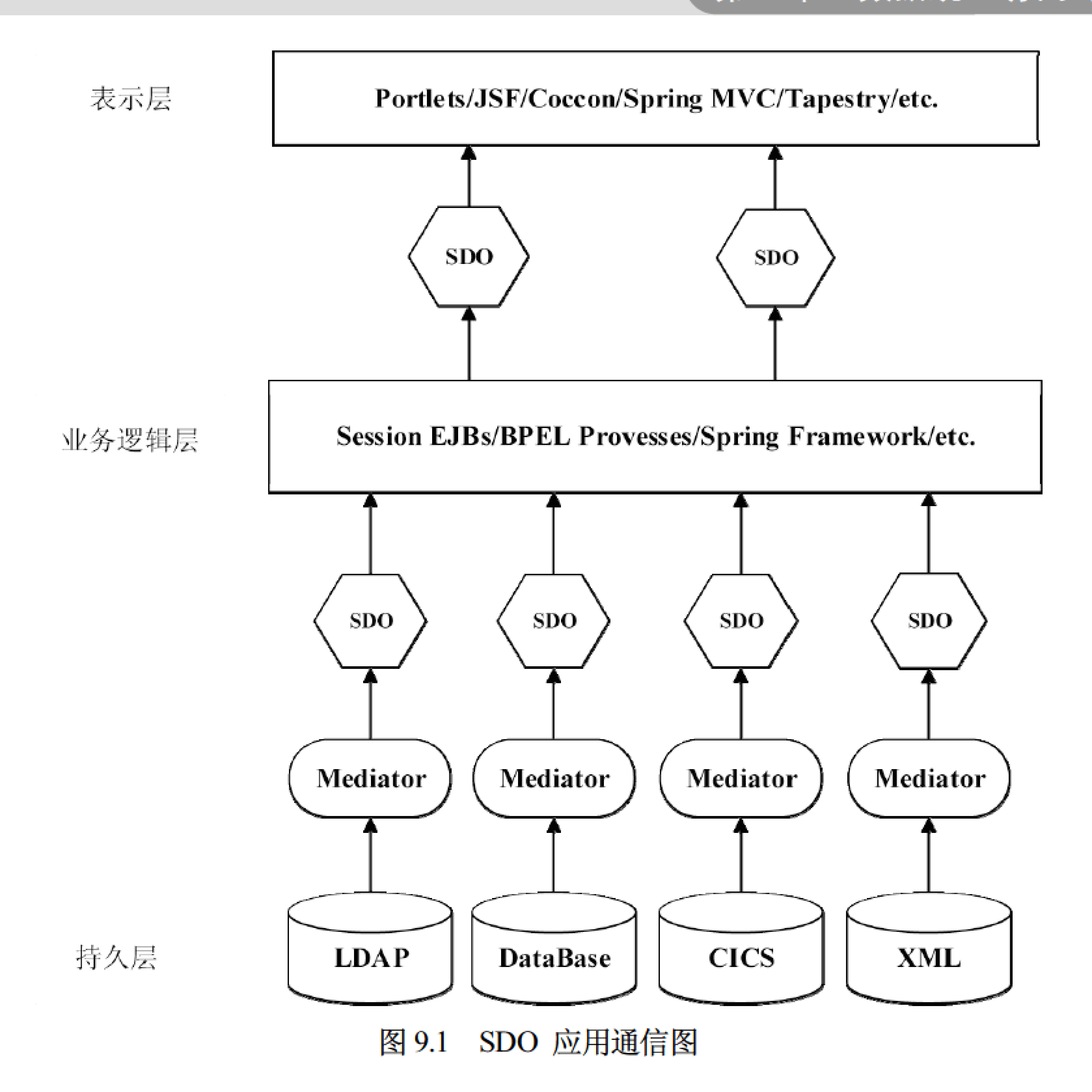


图 9.1　SDO 应用通信图

SDO 解决了异构数据的兼容性的问题，提出了一个简单并统一的模式供服务处理其相关的数据。开发人员可以用 SDO 统一其数据访问和处理模式，即使这些数据来源于异构数据源——关系数据库、XML 数据、Web 服务或者是企业信息系统。

SDO 通过以下手段简化数据编程：

- 统一了不同数据源类型的数据编程，屏蔽了数据库底层的差异，对异构数据可以通过相同接口来调用，使应用把精力放在数据的逻辑结构，而不是把物理差异也考虑进去；使数据库迁移及版本升级变得很容易，无需修改数据操作逻辑。
- 动态数据类型，传统静态数据类型如 JavaBean 对操作数据要完全清楚，然后通过 get 和 set 方法来操作该数据类型，而对应无法预先知道的数据，SDO 则更适合，SDO 根据实际数据需求，可以动态地组合数据（可以通过 XSD 配置，也可以通过代码），添加、修改属性。
- 提供了对通用应用程序模式的健壮支持，使应用程序、工具和框架更容易查询、读取、绑定、更新和检查数据；变更摘要使数据操作变得更得心应手。

上述介绍的服务数据对象技术为异构数据的统一访问提供了理论基础，模型驱动体系架构（Model Driven Architecture，MDA）技术则为异构数据的转换提供了可能。

软件技术发展的过程中，软件开发者开发出具有较高层次的抽象程序设计语言，使得软件能够实现更大程度的复用。在这些理论和技术的支持下，2001 年对象管理组织（Object Management

Group，OMG）提出以模型为中心的软件开发框架标准——模型驱动体系结构（Model Driven Architecture，MDA），使得软件抽象和复用技术达到了一个新的高度。

MDA 将数据业务和应用逻辑与底层平台技术分离开来。通过使用 UML 以及其他的 OMG 建模标准，来表达应用程序或者集成系统的业务功能和行为，得到的平台无关模型可以通过 MDA 实现到各种平台上，如 Web Services、.NET、CORBA、J2EE 等。这些平台无关模型将应用的业务功能与行为同实现它们的技术特定的代码分离开来，如图 9.2 所示。

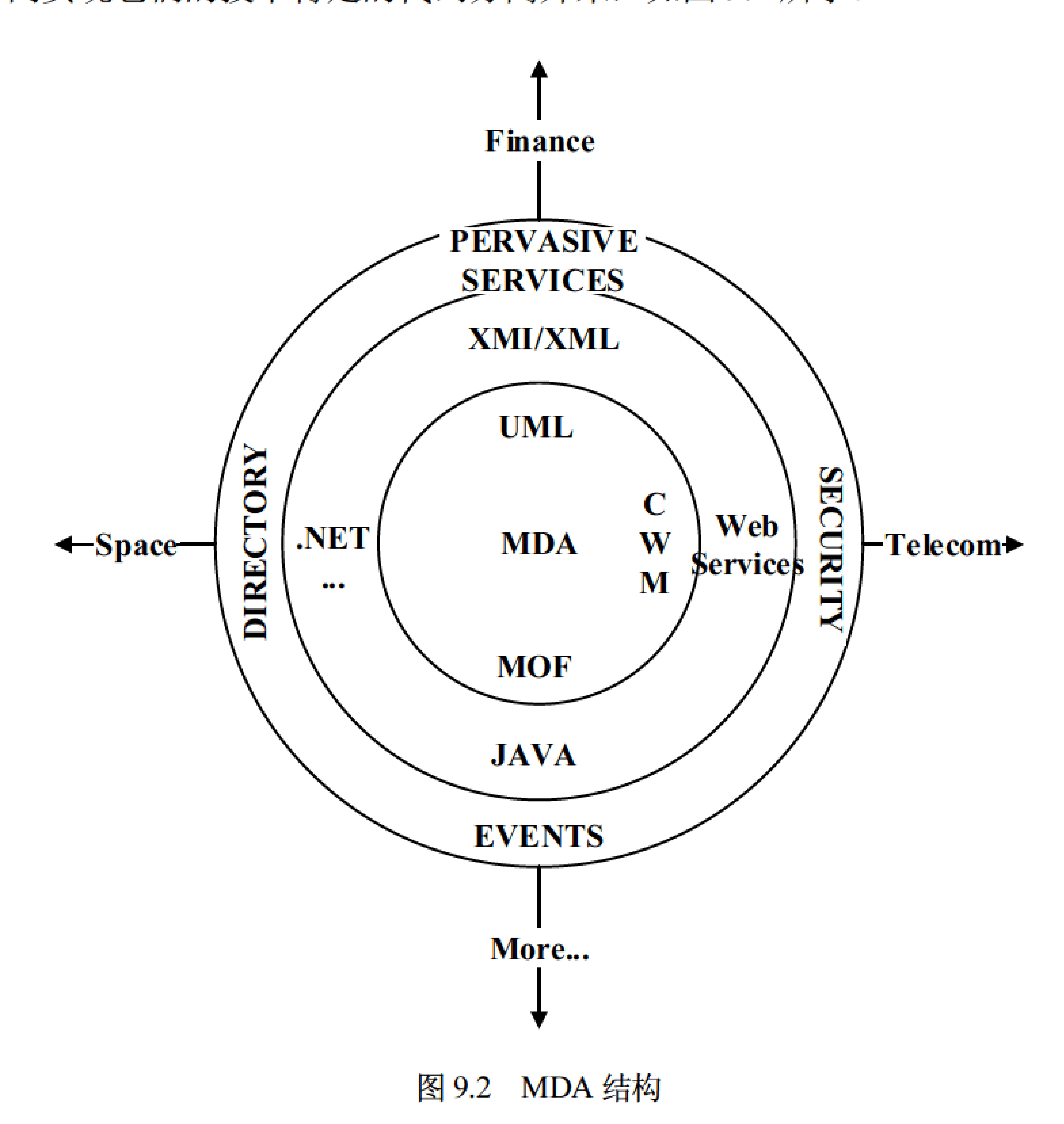


图 9.2　MDA 结构

图 9.2 中最内环是 MDA 的核心技术：MOF（Meta Object Facility，元对象设施）、 CWM（Common Warehouse Meta-model，公共数据仓库元模型）和 UML。 MDA 的主要工作就是要把基于这些技术建立的 PIM（Platform Independent Model，平台独立模型）转换到不同的中间件平台上，得到对应的 PSM（Platform Specific Model，平台相关模型）。中间环上给出的是目前主要针对的实现平台： CORBA、XML、JAVA、Web Services 和.NET 。显然，随着技术的发展，这个列表将不断扩充。最外环是 MDA 提供的公共服务，如事务（Transactions）等，向外发散的箭头是指 MDA 在不同垂直领域的应用，如电子商务、电信和制造业等。基于 MDA 应用基础为数据的转换提供了一种新的方式。

9.2 数据统一访问需求分析与总体设计

9.2.1 功能性需求分析

通过对应用背景的描述分析可以提取出数据统一访问和转换的功能性需求，其功能性需求的主要用例图如图 9.3 所示。

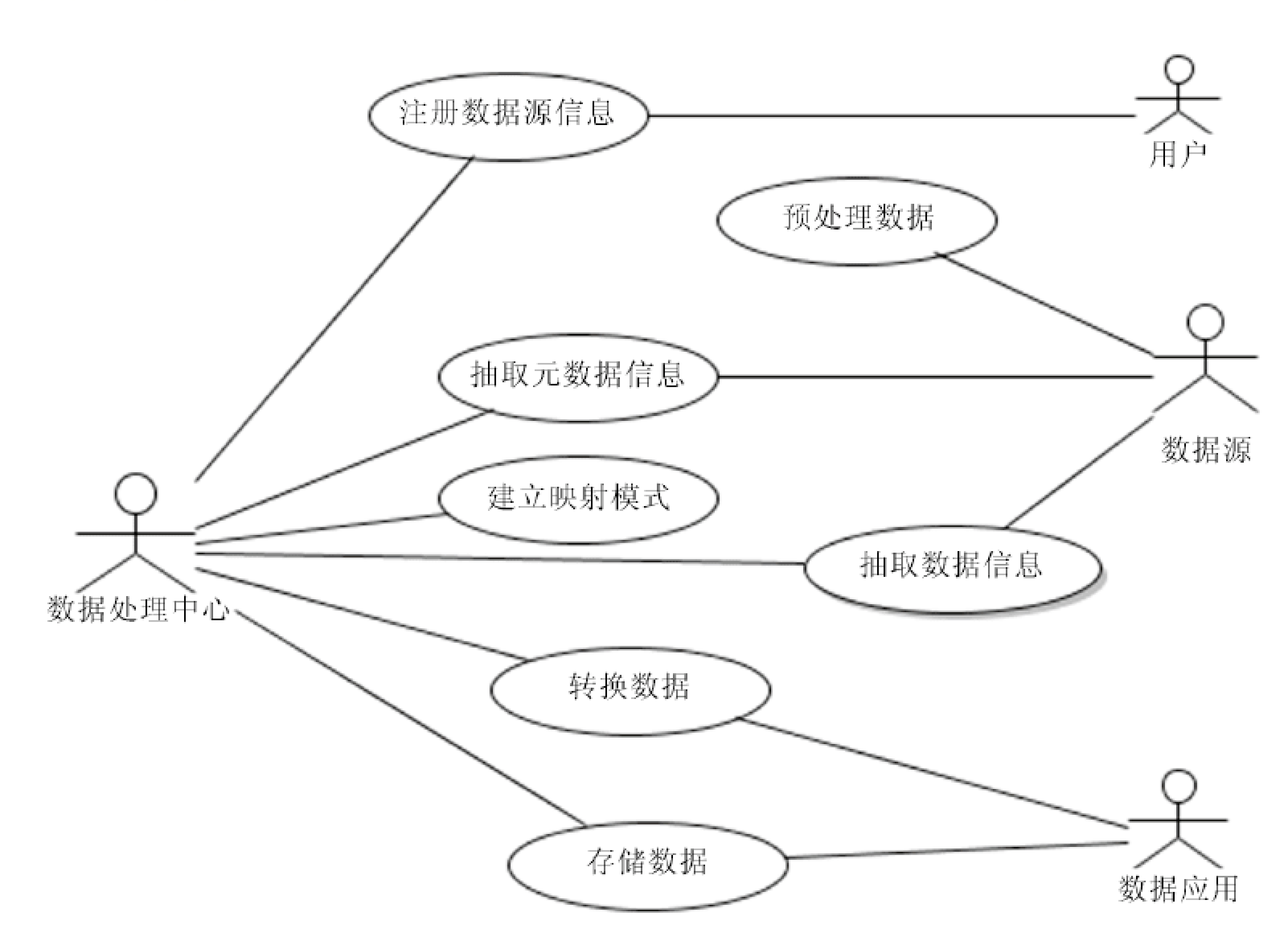


图 9.3　数据统一访问和转换用例图

从图 9.3 中可得出主要的功能有：用户注册数据源信息到数据处理中心、异构数据源对数据源中存储的全部数据进行预处理、数据处理中心抽取相关异构数据源的元数据信息、数据处理中心依据元数据信息建立映射模式、数据处理中心抽取数据源中的数据信息、数据处理中心对异构数据做灵活转换以及对转换后数据的存储等。数据应用主要是对数据处理中心对转换后的数据应用或者对存储数据的特定应用。

由于在异构数据源中的数据量可能是巨大的，因此在抽取数据信息和转换数据存储数据的过程中主要借助于服务数据对象编程技术 Hadoop 平台、分布式 MapReduce 计算框架和 HBase 存储等相关技术来达到高效、快速、准确的运算和存储操作。

1. 数据预处理

数据预处理功能的目的是保证数据的基本质量，为数据的抽取、转换、存储等提供基础服务。数据源处理工作主要在数据源本地来完成，通过对数据的清洗，数据的过滤、去重以及数据修正等技术手段来保证数据的基本质量以使其能够满足数据分析抽取等过程的统一处理。

预处理数据详细用例规约如表 9.1 所示。

表 9.1　预处理数据详细用例规约

用例编号	UC-1	
用例名称	预处理数据	
参与者	数据源	
描述	该用例描述了数据源对数据源数据做预处理的过程	
用例典型事件流	参与者动作	系统响应
	Step1：获取数据源数据信息（最大值、最小值、记录数、空值数目等）	
		Step2：统计并返回记录数
		Step3：统计并返回空值数目
		Step4：分析并返回最大值
		Step5：分析并返回最小值
	Step6：依据返回的分析统计信息审核校验数据信息	
可选事件流	Step2：统计记录数返回为零（记录为空值）时，不再进行最大值、最小值、空值等的统计 Step6：因为 Step2 中的统计记录数为空值，因此不再需要对数据信息进行审核校验	
前置条件	数据源数据信息记录数不为空值	
后置条件	可以对数据源数据信息按照统一的规则进行分析处理等操作	
假设	无	

2. 注册数据源信息

注册数据源信息功能允许用户将需要的异构数据源信息（数据源访问信息，如数据库的访问地址、端口、数据库名、用户名和密码以及权限等）注册到数据处理中心。数据处理中心得到数据源信息后可以随时访问数据源以获取数据源的数据信息。

注册数据源信息详细用例规约如表 9.2 所示。

表 9.2　注册数据源信息详细用例规约

用例编号	UC-2	
用例名称	注册数据源信息	
参与者	用户、数据处理中心	
描述	该用例描述了用户向数据处理中心注册数据源信息的过程	
用例典型事件流	参与者动作	系统响应
	Step1：注册数据源信息	
		Step2：提示输入数据源的信息
	Step3：输入数据源的详细信息	
		Step4：接收输入的数据源信息并校验验证数据的正确性
		Step5：测试数据源的访问并返回注册成功提示信息
	Step6：注册完成	
可选事件流	Step4：输入的数据源信息校验失败，返回到 Step3 提示用户重新输入数据源的信息 Step5：测试数据源访问失败，提示用户检查数据源信息并重新测试或修改数据源信息	
前置条件	数据源可访问，数据处理中心可连接	
后置条件	数据处理中心存储数据源信息	
假设	无	

3. 抽取数据源元数据信息

抽取数据源元数据信息功能允许数据处理中心抽取异构数据源的元数据信息，这些信息包含数据库名、数据库表名、属性（类型名、格式、约束等）以及主键、外键等的描述，标准元数据通常被用来访问分布式异构数据源。可以通过服务数据对象（Service Data Object，SDO）、数据访问服务（Data Access Service，DAS）API 读取数据库元数据（Metadata）信息，提取对应的异构资源数据库的所有表信息、视图以及相关的规则和语义约束（如主外键、唯一性约束、默认值等）信息。

抽取数据源元数据信息的详细用例规约如表 9.3 所示。

表 9.3　抽取数据源元数据信息详细用例规约

用例编号	UC-3	
用例名称	抽取数据源元数据信息	
参与者	数据处理中心、数据源	
描述	该用例描述了数据处理中心抽取数据源元数据信息的过程	
用例典型事件流	参与者动作	系统响应
	Step1：数据处理中心获取数据源元数据信息，如数据库名、数据库表名、属性（类型名、格式、约束等）以及主键、外键等	
		Step2：依据数据处理中心存储的数据源访问信息连接数据源
		Step3：获取并返回数据库名
		Step4：获取并返回数据库表名
		Step5：获取并返回属性详细信息
		Step6：获取并返回主键外键等其他约束信息
	Step7：数据处理中心存储数据源元数据	
可选事件流	Step2：数据源连接失败，尝试重新连接 Step6：数据处理中心存储数据源数据完成异常时提示抽取异常	
前置条件	数据处理中心已获取数据源访问信息（数据源已注册）	
后置条件	访问完成后断开数据源连接	
假设	无	

4. 建立映射模式

建立映射模式功能主要是为了解决数据转换中各异构数据源中数据模型的异构性，使用以局部模式实体存储数据源局部模式以实现映射，既最大限度地保留原数据源的各种信息，又保证这种映射没有过多的冗余信息。数据源的全局模式包括数据源元数据全局模式和领域知识元数据全局模式两个方面。

建立映射模式的详细用例规约如表 9.4 所示。

表 9.4　建立映射模式详细用例规约

用例编号	UC-4	
用例名称	建立映射模式	
参与者	数据处理中心	
描述	该用例描述了数据处理中心依据抽取的数据源元数据信息建立映射模式的过程	
用例典型事件流	参与者动作	系统响应
	Step1：数据处理中心建立模式映射	
	Step2：数据处理中心读取存储的数据源元数据	
		Step3：获取并返回数据源元数据等信息
	Step4：数据处理中心依据返回的数据源元信息建立映射模式	
可选事件流	Step3：数据源元数据信息获取失败 Step4：数据处理中心建立映射模式失败	
前置条件	数据处理中心已抽取数据源元数据信息	
后置条件	无	
假设	无	

5. 抽取数据源数据信息

抽取数据源数据信息功能主要是获取数据源中存储的数据信息以供后续的转换和应用等。抽取数据源数据信息可以通过服务数据对象（Service Data Object，SDO）、数据访问服务（Data Access Service，DAS）API 读取数据源中的数据信息。

抽取数据源数据信息的详细用例规约如表 9.5 所示。

表 9.5　抽取数据源数据信息详细用例规约

用例编号	UC-5	
用例名称	抽取数据源数据信息	
参与者	数据处理中心、数据源	
描述	该用例描述了数据处理中心抽取数据源数据信息的过程	
用例典型事件流	参与者动作	系统响应
	Step1：数据处理中心获取数据源数据信息	
		Step2：依据数据处理中心存储的数据源访问信息连接数据源
		Step3：获取并返回数据源数据信息
	Step4：数据处理中心读取数据源数据信息	
可选事件流	Step2：数据源连接失败，尝试重新连接 Step4：数据处理中心读取数据源数据异常时提示抽取数据源数据异常	
前置条件	数据处理中心已获取数据源访问信息（数据源已注册）	
后置条件	访问完成后断开数据源连接	
假设	无	

6. 转换数据

转换数据功能是最重要的一个功能，主要是用于在异构数据源数据间的转换。转换过程中如果异构数据源间的数据表达方式一致，则在转换过程中直接把原数据源的数据复制到目标数据源以供后续的应用，否则按照预先定义好的原数据源数据表达方式到目标数据源数据的表达方式的转换过程来实现转换，也可以像关系数据库中的存储过程一样，由用户实现转换方式并注册到数据处理中心，然后主动调用来转换。

转换数据的详细用例规约如表 9.6 所示。

表 9.6　转换数据详细用例规约

用例编号	UC-6	
用例名称	转换数据	
参与者	数据处理中心、数据应用	
描述	该用例描述了数据处理中心按照映射模式中的信息对原数据源数据转换的过程	
用例典型事件流	参与者动作	系统响应
	Step1：数据处理中心获取原数据源数据信息	
		Step2：读取并返回数据处理中心请求的原数据源数据信息
		Step3：扫描映射模式，比较源目标数据源间的一致性
	Step4：选择数据转换方式	
		Step5：按照 Step4 中选择的转换方式实现数据的转换并返回转换后的数据
	Step6：数据处理中心应用或存储转换后的数据	
可选事件流	Step2：数据源数据读取失败，转换失败 Step4：数据处理中心不存在符合要求的转换操作时，提示用户注册转换过程或退出	
前置条件	数据处理中心已获取数据源访问信息（数据源已注册） 原数据源可连接	
后置条件	访问完成后断开数据源连接	
假设	无	

7. 存储数据

存储数据功能主要用来存储抽取的数据源元数据信息以及异构数据源间的数据转换信息等。数据的存储可以借助于 Hadoop 平台下的 HBase 数据库实现。

存储数据的详细用例规约如表 9.7 所示。

表 9.7 存储数据详细用例规约

<table>
<tr><td>用例编号</td><td colspan="2">UC-7</td></tr>
<tr><td>用例名称</td><td colspan="2">存储数据</td></tr>
<tr><td>参与者</td><td colspan="2">数据处理中心、数据应用</td></tr>
<tr><td>描述</td><td colspan="2">该用例描述了数据处理中心存储数据的过程</td></tr>
<tr><td rowspan="6">用例典型事件流</td><td>参与者动作</td><td>系统响应</td></tr>
<tr><td>Step1：数据处理中心提交并存储数据</td><td></td></tr>
<tr><td></td><td>Step2：接收数据处理中心提交的数据并对数据进行有效性验证</td></tr>
<tr><td></td><td>Step3：Step2 校验成功完成后调用 HBase 数据存储接口完成数据的存储</td></tr>
<tr><td></td><td>Step4：存储完成后提示存储成功</td></tr>
<tr><td>Step5：数据处理中心应用或存储数据完成</td><td></td></tr>
<tr><td>可选事件流</td><td colspan="2">Step2：数据有效性校验失败时存储失败并返回
Step3：调用 HBase 数据存储接口存储数据失败时，数据存储失败并返回</td></tr>
<tr><td>前置条件</td><td colspan="2">数据处理中心连接 HBase 数据库成功</td></tr>
<tr><td>后置条件</td><td colspan="2">访问完成后断开 HBase 数据库连接</td></tr>
<tr><td>假设</td><td colspan="2">无</td></tr>
</table>

9.2.2 非功能性需求分析

在数据统一访问以及灵活转换系统中的非功能性需求主要体现在数据质量方面。下面主要从 4 个方面来介绍数据质量的基本要素以及如何评估数据的质量。这 4 个方面主要是完整性、一致性、准确性以及及时性，它们共同构成了数据质量的 4 个基本要素，四要素与数据质量的关系图如图 9.4 所示。

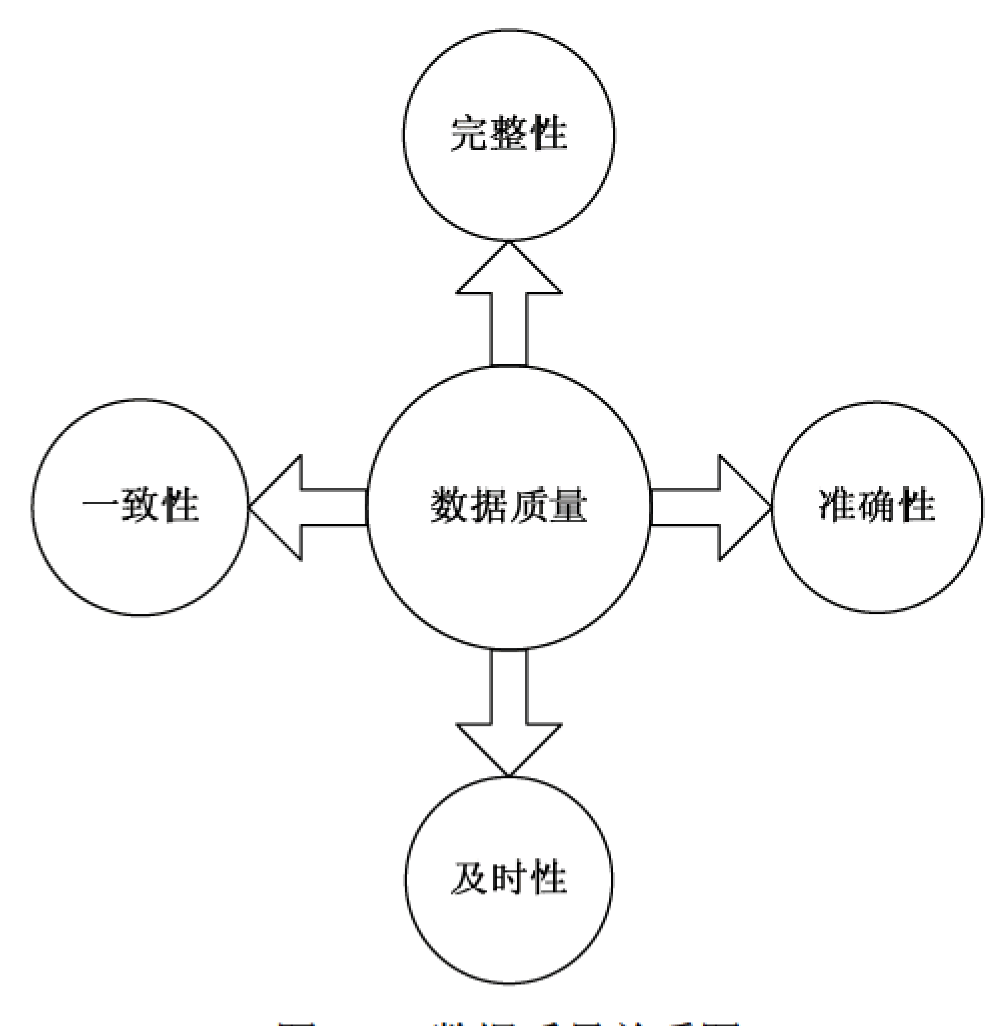


图 9.4 数据质量关系图

1. 完整性

完整性用来描述数据的记录和信息是否完整，是否存在缺失的情况。

数据的缺失主要有记录的缺失和记录中某个字段信息的缺失，两者都会造成统计结果的不准确，所以完整性是数据质量最基础的保障，而对完整性的评估相对比较容易。

2. 一致性

一致性用来描述数据的记录是否符合规范，是否与前后及其他数据集合保持统一。

数据的一致性主要包括数据记录的规范和数据逻辑的一致性。数据记录的规范主要是数据编码和格式的问题，比如网站的用户 ID 是 15 位的数字、商品 ID 是 10 位数字，商品包括 20 个类目、IP 地址一定是用“，”分隔的 4 个 0~255 的数字组成，以及一些定义的数据约束，比如完整性的非空约束、唯一值约束等；数据逻辑性主要是指标统计和计算的一致性，比如 PV>=UV，新用户比例在 0~1 之间等。数据的一致性审核是数据质量审核中比较重要，也是比较复杂的一块。

3. 准确性

准确性用来描述数据中记录的信息和数据是否准确，是否存在异常或者错误的信息。

导致一致性问题的原因可能是数据记录的规则不一，但不一定存在错误；而准确性关注的是数据记录中存在的错误，比如字符型数据的乱码现象也应该归到准确性的考核范畴，另外就是异常的数值，异常大或者异常小的数值，不符合有效性要求的数值，如访问量 Visits 一定是整数、年龄一般在 1~100 之间、转化率一定是 0~1 的值等。对数据准确性的审核有时会遇到困难，因为对于没有明显异常的错误值我们很难发现。

4. 及时性

及时性用来描述数据从产生到可以查看的时间间隔，也叫数据的延时时长。

虽然说分析型数据的实时性要求并不是太高，但并不意味着就没有要求，分析师可以接受当天的数据要第二天才能查看，但如果数据要延时两三天才能出来，或者每周的数据分析报告要两周后才能出来，那么分析的结论可能已经失去时效性，分析师的工作只是徒劳；同时，某些实时分析和决策需要用到小时或者分钟级的数据，这些需求对数据的时效性要求极高，所以及时性也是数据质量的组成要素之一。

9.2.3　总体设计

基于之前对数据统一访问与转换平台的需求分析，得出系统的功能性能等要求，并提出基于服务数据对象（SDO）的数据统一访问与转换平台总体设计方案。

根据对数据转换等相关问题的分析，数据统一访问与转换平台需要满足以下一些基本的要求。

- 统一性。对异构数据源数据进行统一，应该挖掘异构数据源之间业务逻辑或数据结构的内在关系，使得整合后的数据成为建立在一定联系上的整体。

- 完整性。为了满足各种应用处理以及发布的要求，整合后的数据必须保证尽可能的完整性，包括数据完整性和约束完整性。
- 一致性。不同数据源之间可能存在着语义上的区别，整合后的数据应该根据一定的数据转换模式和业务规则进行统一的数据结构和字段语义编码转换。
- 安全性。由于数据源来源不统一，某些数据存在一定的安全性，因此在实现数据共享的同时必须充分保证相关数据的隔离性。
- 准确性。数据统一平台需要能够提供给用户准确的数据。分布式查询系统的正确性依赖于正确的查询分解、局部查询和查询结果汇总。
- 访问透明性。系统必须提供查询的透明性，用户不需要理解系统的结构，使用系统的统一接口就实现正确的查询请求，并由系统透明地返回查询结果。
- 及时性。在大数据时代数据源进行联合查询中最重要的问题。系统必须能够尽量降低对系统资源的占用以及响应时间。

首先来描述数据统一访问与转换平台的系统数据流图，如图 9.5 所示。

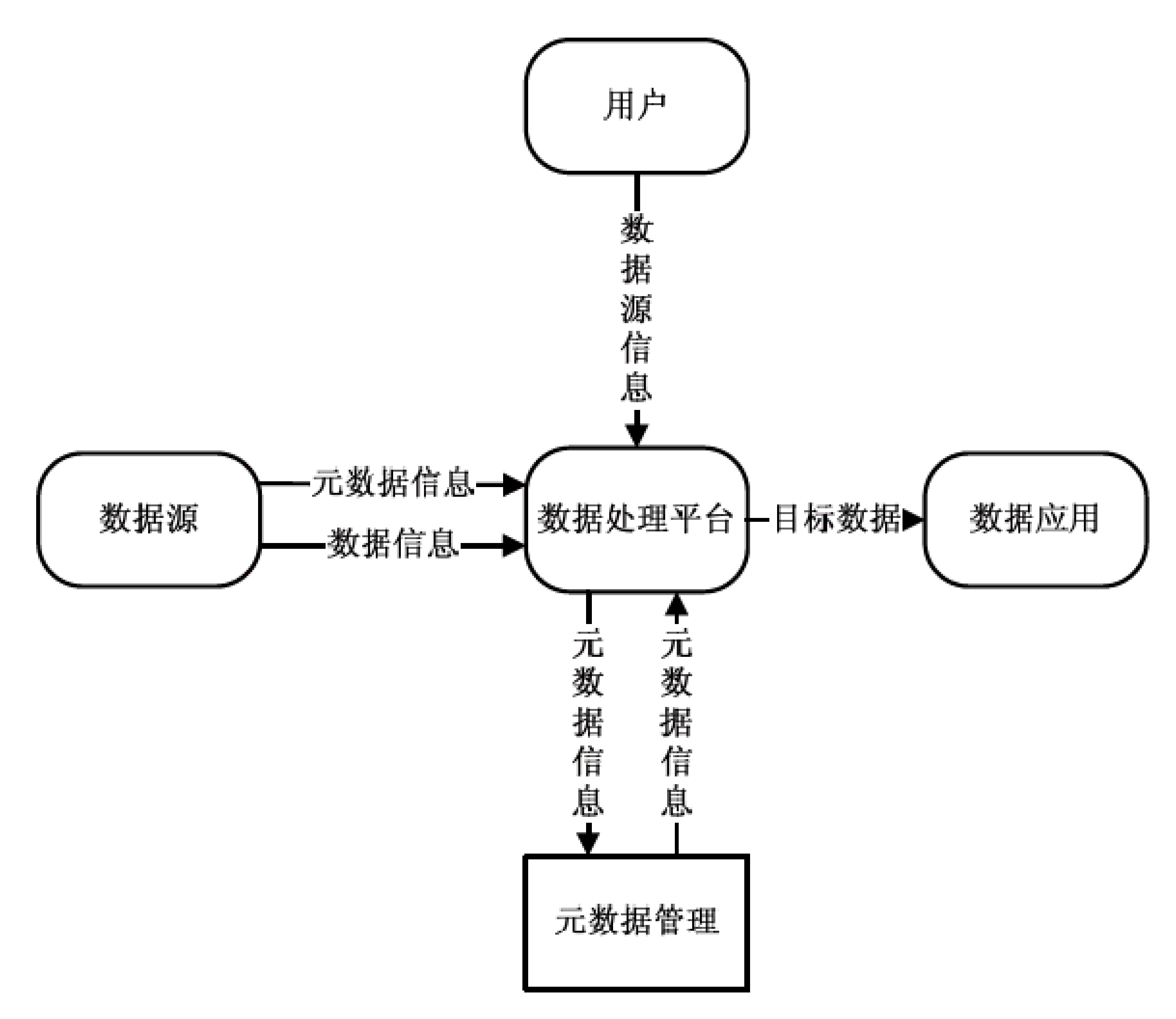


图 9.5　数据统一访问与转换平台数据流图

数据源是数据处理中心的基础，用户往数据处理中心注册数据源信息后，数据处理中心从数据源抽取元数据信息并存储元数据信息，然后通过元数据管理模块创建异构数据源间的映射模式。在用户选择数据转换操作后，在数据处理中心处理数据转换过程中，数据处理中心读取元数据以及相应的映射模式实现异构数据的转换过程。最后数据处理中心将转换后的目标数据返回给用户应用或者将目标数据存储到目标数据库中。

在介绍了数据统一访问与转换的数据流图后，根据前面的要求，对数据统一访问与转换平台的总体系统设计架构图如图 9.6 所示。

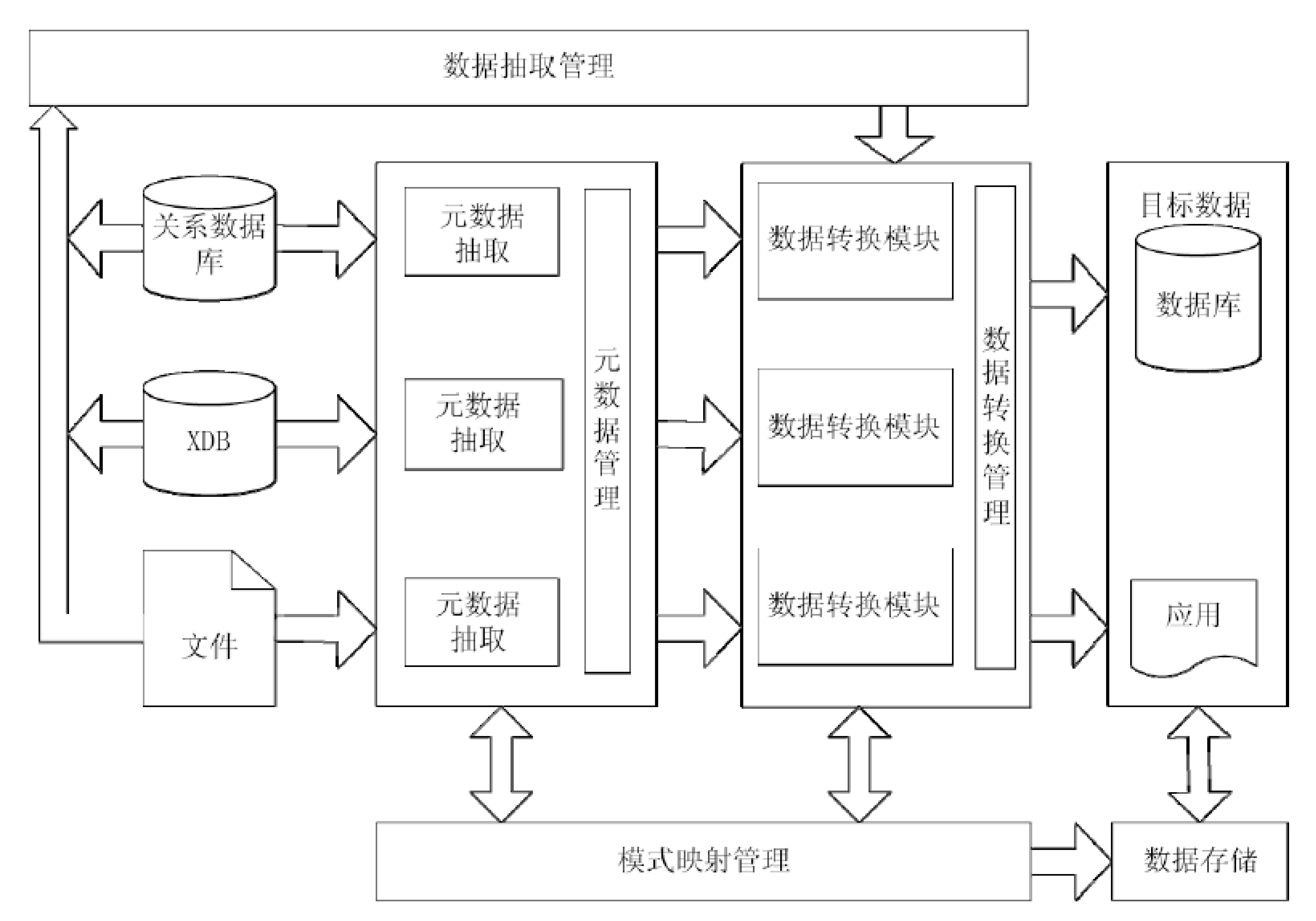


图 9.6　数据统一访问与转换架构图

从数据统一访问与转换平台的系统设计架构图可以看出整体架构由 7 部分组成：数据源部分、元数据抽取管理部分、数据抽取管理、数据转换管理、模式映射管理以及数据存储管理和目标数据应用。

其中元数据抽取管理部分和数据抽取管理构成了数据源访问模块，数据访问模块负责根据所选择的数据源的类型，获取元数据信息，以及运行时与数据源交互完成数据抽取的工作。在源元数据信息抽取和数据源数据信息抽取过程中主要使用服务数据对象（SDO）和数据访问服务（DAS）来实现上述要求。

模式映射管理和数据转换管理组成数据转换管理模块，数据转换管理模块中用户需要定义各转换节点的转换规则，创建任务的工作流，依据模式映射构建从源到目标的字段映射转换等，再将这些映射规则（元数据）存储在元数据管理模块中；用户在执行任务时，系统从元数据管理模块中查询转换映射规则并完成数据的转换。该模块中使用的技术主要有分布式云计算 Hadoop 平台、HBase 数据库技术以及分布式 MapReduce 计算框架。

元数据管理与模式映射管理组成了模式管理模块，模式管理模块中元数据管理模块主要负责元数据的抽取和解析，这里的元数据主要包括数据源信息描述、目标信息描述，模式映射管理主要是依据元数据管理模块提供的元数据信息创建转换和映射规则信息描述等。

数据存储管理主要负责模式映射管理模块创建的映射模式存储以及转换规则、转换中间数据的存储等。

9.3 数据统一访问与转换关键技术

9.3.1 SDO 编程技术

SDO 采用离线数据图的设计理念。数据图是一组树型结构或者图型结构的数据对象。离线的访问方式是指客户端从数据源提取并构建数据图，然后在应用中操作数据图，并在变更摘要（Change Summary）中记录相应的数据操作，在动作结束后由数据访问服务（Data Access Service）批量地将相应的改变反映回数据源，其中数据源可以是异构的，并不仅仅限于关系数据库。SDO 的基本结构如图 9.7 所示。

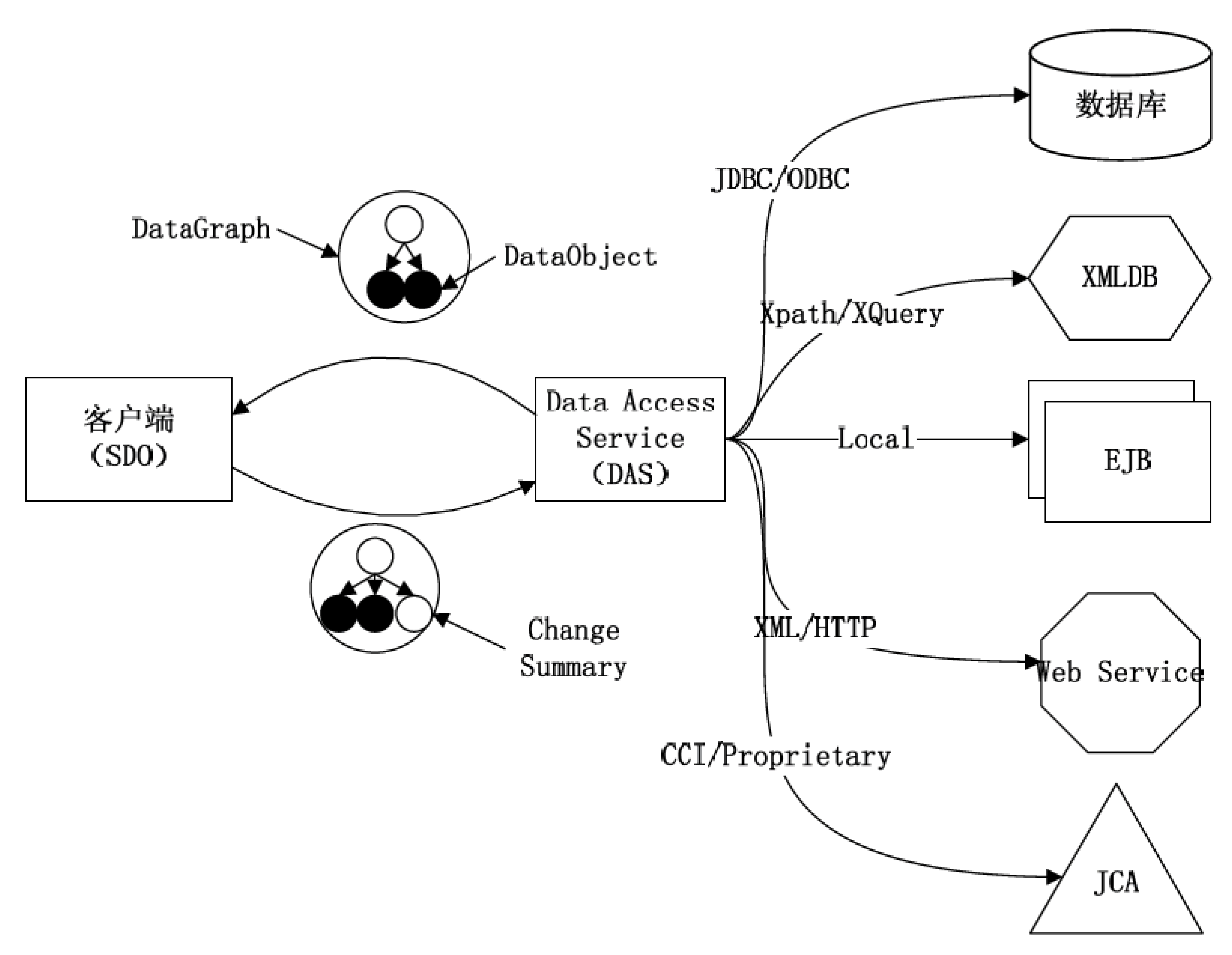


图 9.7　SDO 的基本结构

SDO 的数据表现形式基于数据对象（Data Object）和数据图（Data Graph）的概念，其封装形式与 Java 类和 XML 有水到渠成的映射关系。同时，SDO 提供了丰富的数据操作接口——动态接口和静态接口，还可以用 XPath 来直接访问相应的数据对象属性。

如图 9.8 所示，SDO 有下面一些主要部分。

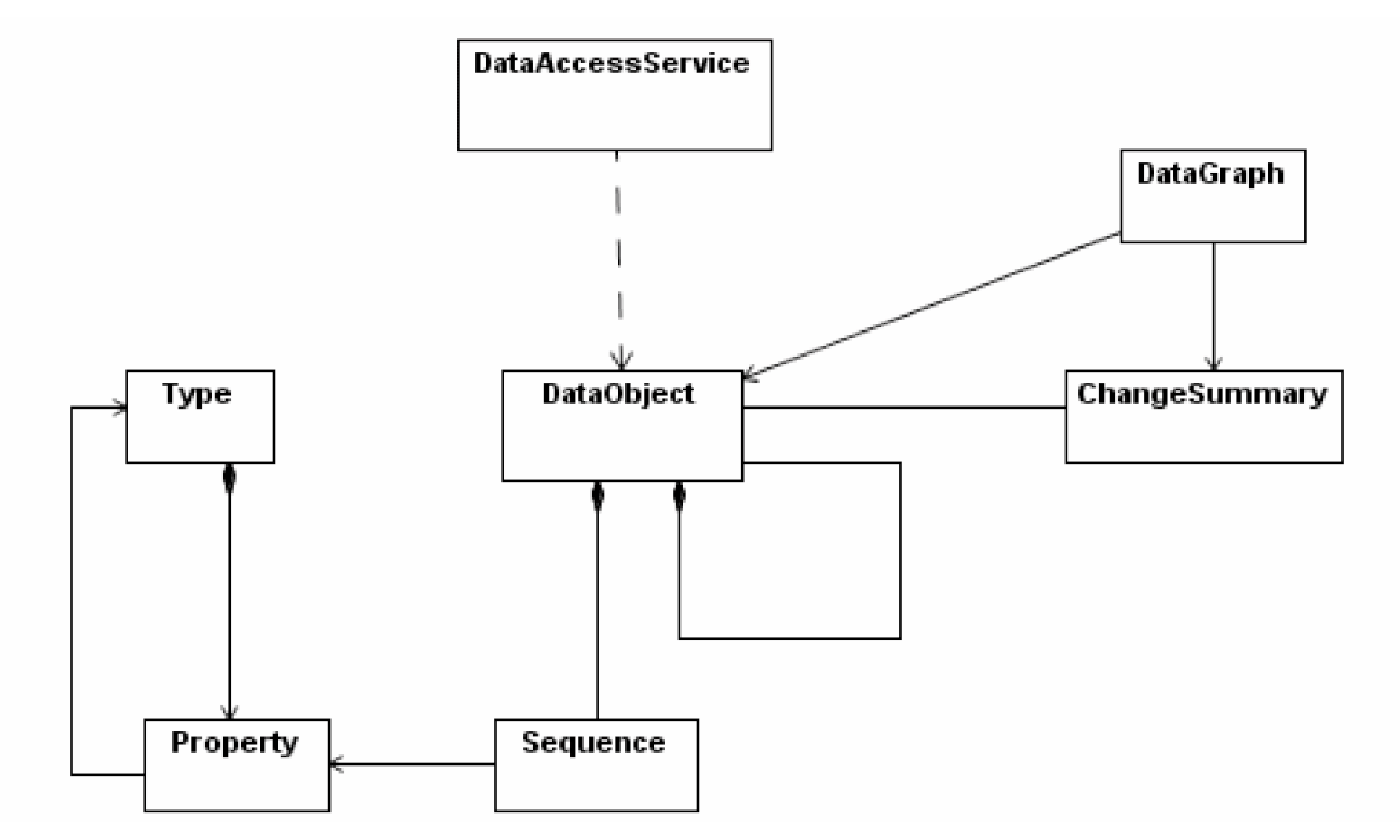


图 9.8　SDO 内部组成关系

1. SDO 的基本概念

（1）数据图（Data Graph）

数据图是一组提供组件之间或层之间的传输单元的数据，数据图是一个描述数据的分层结构，数据图记录所有对数据的更改，包括新的数据对象、被更改的数据对象以及被移除的数据对象称作变更摘要（Change Summary）的结构。变更摘要记录了数据图中所有数据对象的历史更改信息。此外，由于数据图是由数据对象组成的，因此它是可序列化的。数据图结构如图 9.9 所示。

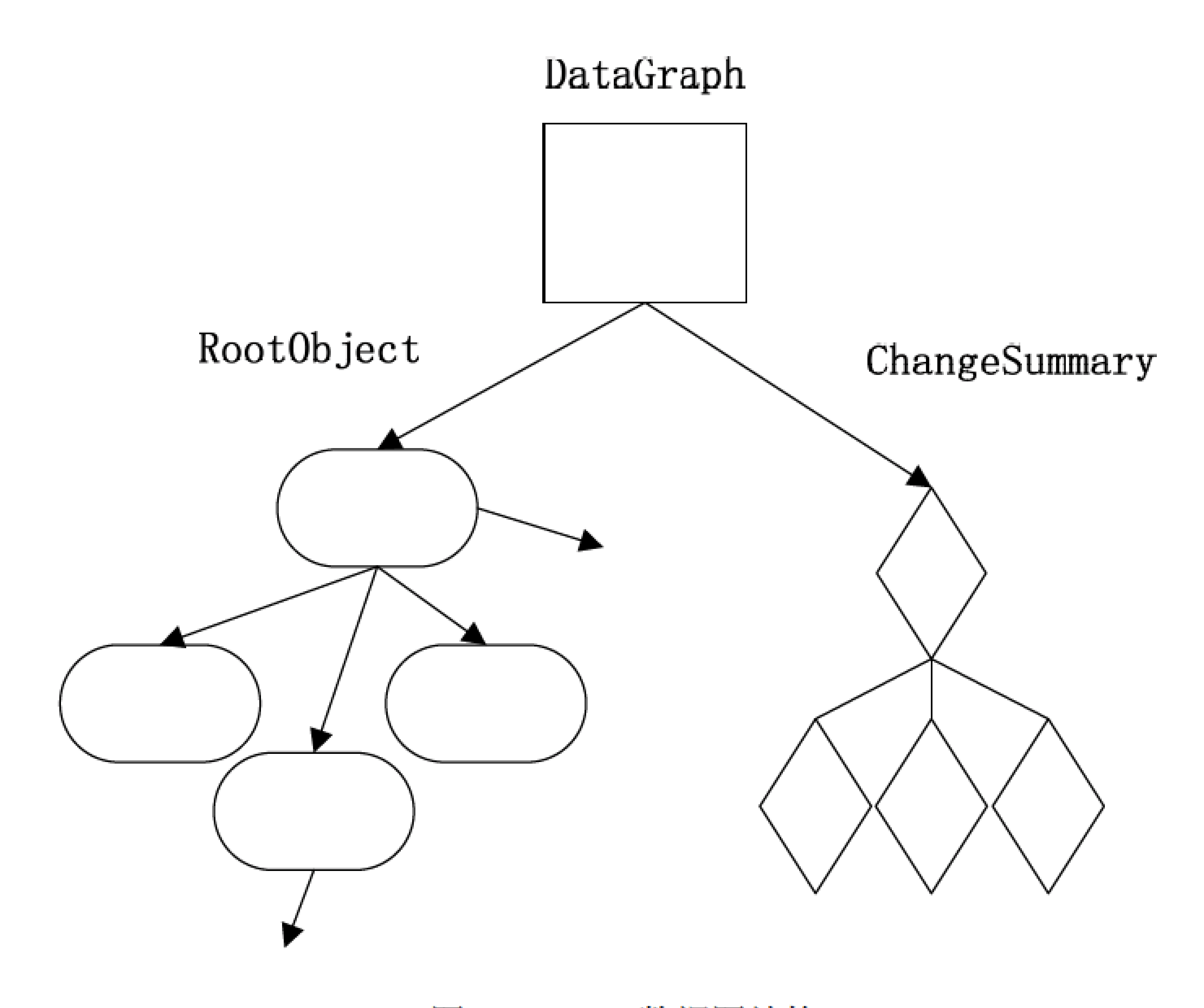


图 9.9　SDO 数据图结构

数据图由 DAS 生成，供 SDO 客户使用。修改后，数据图被回传给 DAS 更新数据源。SDO

客户可以遍历数据图，读取和修改数据图中的数据对象。当在应用程序组件（比如服务调用期间的 Web 服务请求者和提供者）之间进行传输、组件和 DAS 之间进行传输（或者保存到磁盘）的时候，数据图被序列化为 XML。SDO 规范提供了序列化的 XML Schema。

数据访问服务（Data Access Service，DAS）是 SDO 离线访问模式的一个不可或缺的部分，尽管现在还不是 SDO 规范中的一部分。数据访问服务可以从后端数据源建立数据图和数据对象，并且将其存回相应的数据源。DAS 支持基于变更摘要的增量更新。

根据 SDO 的思路，SDO 将通过 SDO 从各种数据源，或者后端关系数据库，或者 XML 文件等得到一个 Data Graph 传到前端。因为 Data Graph 里面有一个根数据对象，由于可以从根数据对象得到所有其他的数据对象，这样从一个 Data Graph 对象就可以得到所有的数据对象了。

（2）数据对象

数据对象是 SDO 的基本组件。简单地说，它是由属性的键/值对组成的，每个值都可以是原始的数据类型，或者是另一个数据对象。数据对象提供了易于使用的创建（createDataObject）和删除（delete）方法，获得自身类型（实例类、名称、属性和名称空间）的反射方法。数据对象都链接在一起，包含在数据图中。数据对象是可序列化的。

通常，人们使用传统的 Java 对象（POJO 或 Java beans）或是传统的 Java 接口（POJI）来以一种持久性-机制-中立的风格表示数据（不久将更多地用于关系型和 XML 数据）。举例来说，人们为了使用 POJO 普遍会构造“数据传输对象”。我们称 Java bean 类型的 API 为“静态的”，因为预先定义好的具有一系列属性（或 getter/setter 方法）的数据类型已经存在了。然而，静态数据 API 并不总能执行，因为有时 Java 类甚至还并不存在。在许多动态查询中，返回数据的形式并不是已知的预先类型，这样就不能将数据填写到已经存在的 Java 类中。另外，数据结构是可扩展的；例如，对于 XML 数据，在剖析它之前，通常不知道它的精确类型（假定它的 XML 模式结构是可扩展的）。这就是 SDO 数据对象接口的便利之处：它提供了“动态的”数据 API。当您需要产生一个能支持包括动态查询、未知数据类型和可括展模式等情况的通用框架时，有一个动态的数据 API 会更加有用。

（3）变更摘要

变更摘要包含在数据图中，表示对 DAS 返回的数据图的修改。变更摘要最初是空的，随着数据图的变化逐渐填充。在后台更新时，DAS 使用变更摘要将修改应用于数据源。变更摘要提供了数据图中被修改的属性（包括原来的值）、新增和删除的数据对象的列表，从而使 DAS 以递增方式高效地更新数据源。只有当变更摘要日志功能被激活时，才会将信息添加到数据图的变更摘要中。变更摘要提供了让 DAS 打开和关闭日志功能的方法。

（4）变更摘要接口

变更摘要接口提供了下列方法：检查日志记录的状态，打开、关闭日志记录，取消自日志开始记录起所有在日志中的变化，返回根数据对象和数据图返回已被修改、创建和删除的数据对象。

标识发生了什么变化（修改、创建、删除）为被改变和被删除的数据对象返回旧值取出过去值时，过去值保存在以 ChangeSummary.Setting 作为接口的实例中，如下所示。

```
public interface Setting
{
Object getValue();                    //得到对应属性的值
Property getProperty();               //得到属性
boolean isSet();                      //属性是否被赋值
}
```

变更摘要（Change Summary）中的 API 如下所示。

```
public interface ChangeSummary
{
void beginLogging();                  //启动记录，开始跟踪变化
void endLogging();                    //终止记录
boolean isLogging();                  //判断记录功能是否启动
void undoChanges();                   //消除所有的改变
DataGraph getDataGraph();             //得到修改记录的数据图
DataObject getRootObject();           //得到根数据对象
List /*DataObject*/ getChangedDataObjects();
boolean isCreated(DataObject dataObject);
boolean isDeleted(DataObject dataObject);
boolean isModified(DataObject dataObject);
DataObject getOldContainer(DataObject dataObject);
Property getOldContainmentProperty(DataObject dataObject);
Sequence getOldSequence(DataObject dataObject);
Setting getOldValue(DataObject DataObject, Property property);
List /*Setting*/ getOldValues(DataObject dataObject);
}
```

（5）属性

数据对象用一系列属性保存其内容。每个属性都有一个类型，该类型既可以是基本类型（如 int）也可以是通用数据类型（如 Date），如果引用的话，还可以是其他数据对象类型。每个数据对象都为属性提供了访问和设置方法（getter 和 setter）。这些访问器方法有不同的重载版本，可以通过传递属性名（String）、编号（int）或者属性元对象本身来访问属性。String 访问器还允许使用类 XPath 的语法访问属性。

属性的主要作用是将 DataObject 和下级的 DataObject 关联起来，或者将 DataObject 和其所包含的数据关联起来。获取数据对象属性（Property），可以通过 DataObject.getInstanceProperty()方法得到这个 DataObject 范围内属性的 List，然后遍历 List 就可以得到 Property。如果知道属性的名字，可以直接通过 getInstanceProperty 来调用。

可以通过属性得到下面的 DataObject 或者值。

首先需要判断 isMany，它对应 xsd 文件里面的如下内容：

```
<xsd:element name="user" type=" userType" minOccurs="0" maxOccurs="unbounded"/>
```

如果 isMany 返回 true，那么需要使用 DataObject.getList（property）的方法来得到下面 DataObject 的 List；如果 isMany 返回 false，那么需要 getDataObject.getDataObject（Property）的方法来得到单个 DataObject。

属性的 API 如下所示。

- getName() 返回属性名。
- getType() 返回属性类型。
- isMany() 如果该属性是多值则返回 true，否则返回 false。
- isContainment() 如果属性为一个包含引用返回 true，对数据类型属性总是 false
- isReadOnly() 如果属性值无法通过 SDO API 进行修改则返回 true。
- getContainingType() 返回声明此属性的类型。
- getAliasNames() 返回该属性的别名列表。
- getOpposite() 如果该属性为双向的，则返回对应属性，否则返回 null。
- getDefault()（以对象类型）返回默认值。
- isNullable() 如果属性的实例可以设置为 null 则返回 true。
- isOpenContent() 如果该属性可设置为开放内容则返回 true。
- getInstanceProperties() 返回一个该属性可用的实例属性的只读列表。
- Get(Property property) 返回该属性的被指定实例属性的值。

（6）类型

SDO 数据类型有用户的数据类型（可能是一个复杂数据类型 complexType，也可能是一个简单类型 simpleType）和 SDO 本身支持的基本数据类型。数据类型就是一个数据对象的描述配置，当读入具体数据时就可以初始化一个数据对象（DataObject）。

SDO 的类型可以和编程语言中类型的概念或者数据建模语言中类型的概念相类比。具体如表 9.8 所示。

表 9.8　SDO 相关类型对比

SDO	类型	属性
Java、C++或 UML	class	class 中的域
XML schema	Complex 和 Simple 类型	Element 和 Attribute
C Struct	C Struct	Struct 的域
关系数据库	Table	Column

类型可以通过 getType()接口获取。从数据对象可以得到该数据对象的类型，只能有一个类型，因此实例化一个数据对象事实上是对一个类型的实现，从数据对象得到类型的 API 为 DataObject.getType()。如果知道了一个数据对象的属性，也就可以从属性直接得到对应该属性的

类型。因为实例化一个属性时，事实上也是对该属性类型的一个实现。从属性得到类型的 API 为：Property.getType()。

类型的 API 如下。

- getName() 返回类型的名字。
- GetURI() 返回类型的 URI。
- getInstanceClass() 返回用来实现 SDO 类型的类。
- isInstance(Object object) 如果指定的对象是该类型的一个实例，则此方法返 true。
- isDataType() 如果该类型指定了数据类型则这个方法返回 true，为数据对象则返回 false。
- isSequenced() 如果该类型指定了有序的数据对象，则此方法返回 true。当方法为 true 时，一个数据对象可以返回一个序列。
- isOpen() 如果该类型允许开放内容，则返回 true。否则返回 false。
- dataObject.getInstanceProperties() 必须与该类型的任何一个数据对象的 dataObject.getType().getProperties()返回相同的值。
- isAbstract() 如果这个类型是抽象类型则返回 true，说明该类型是不可实例化的。抽象类型不能用在数据对象或数据工厂的 create 方法中。抽象类型一般作为那些可实例化类型的基类。
- getBaseTypes() 返回该类型的基类列表。如果没有任何基类，则列表为空。XSD 的 <extension>、<restriction>和 Java 的 extends 关键字映射到此基类列表。
- getAliasNames() 返回该类型的别名列表。如果没有任何别名，则列表为空。
- getProperties() 返回一个该类型的所有属性的只读列表，包括那些在基类型中声明的属性。
- getDeclaredProperties() 返回一个在该类型中定义的所有属性的只读列表，不包括那些在基类型中声明的属性。
- getProperty(String propertyName) 返回指定的属性，如果不存在给定名字的属性，则返回 null。
- getInstanceProperties() 返回一个该类型可用的实例属性的只读列表。
- Get(Property property) 返回该类型的指定实例属性的值。

（7）序列

序列概念的引入是为了保持数据对象中属性的顺序。在数据对象中，如果某个属性被标明为多值属性，则其多个值的顺序是由数据对象维护的。但是，这一顺序并没有在不同属性之间维护。如果一个数据对象的类型被定义为顺序，则有：

getType().isSequenced()=true

那么在不同的属性之间也会维护一定的顺序。这一功能可以支持某些半结构化数据的表示。

某个数据对象有两个多值属性，numbers 和 letters。那么对于这样一组值{1,“annotation text”,“A”,2,“B”}，在数据对象中的表示是，属性 numbers 有两个值 1 和 2，属性 letters 有两个值：A 和 B，还有一个字符串“annotation text”。其中，1 和 2 的顺序，A 和 B 的顺序是可以保障的，

但是这组值的整体顺序已然不存在。如果这一数据对象被声明为一个序列，则序列会保持这样的设置：{<numbers,1>,<null, “annotation text” >,<letters，“A” >,<numbers，2>,<letters，“B” >}。

序列的接口如下。

- size()方法返回序列中条目的数量。
- getProperty(int index)访问器返回指定索引位置的属性，如果是非结构化文本条目则返回null。
- getValue(int index)访问器返回指定索引位置的值。
- setValue(int index, Object value)访问器更新指定索引位置的值，并保持序列的位置。
- add()访问器添加到序列的尾端。
- addText(int index, String text)访问器往指定索引位置添加非结构化文本。
- addText(String text) 访问器往序列尾端添加非结构化文本。
- 其他 add(int index)访问器添加到序列中的指定位置并像 java.util.List 一样把此索引位置后的条目上移。
- remove()方法移除指定索引位置的条目，并把此索引位置后的所有条目下移。
- move()方法把条目从一个源索引位置（fromIndex）移动到目的索引位置（toIndex），把 fromIndex 索引后的条目下移，并把 toIndex 索引后的条目上移。

2. SDO 环境的搭建

要构造 SDO 环境，首先需要安装 TUSCANY，可以参考 SCA 的文档。

步骤01 从 http://tuscany.apache.org/sdo-java-download.html 网站上下载 Windows Binary 版本的 SDO：tuscany-sdo-1.0-incubating-beta1-bin.zip。

步骤02 将上述 zip 文件解压，例如解压到 E 盘，则目录结构如图 9.10 所示。

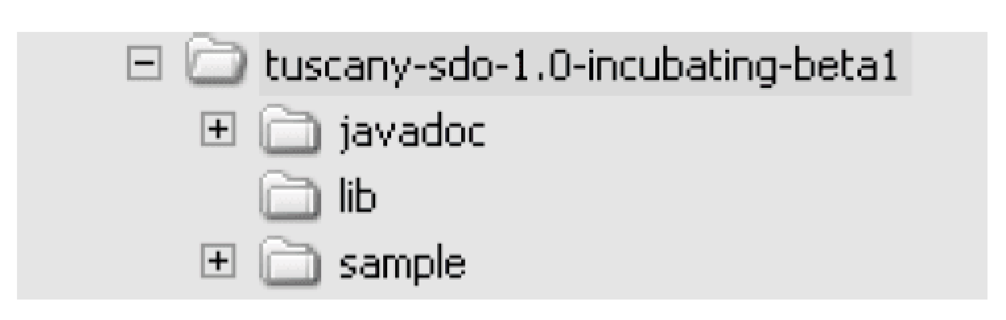


图 9.10　解压目录结构

步骤03 配置 Tuscany。

从菜单栏中选择 Window > Preference ，在 Preferences 对话框中选择 Java > Build Path > User Libraries，单击 New 按钮新建一个 user library。输入 SDO 作为 user library 的名字。单击 Add JARs，将 E:\tuscany-sdo-1.0-incubating-beta1\lib 中所有 JARs 加到该 user library，如图 9.11 所示。

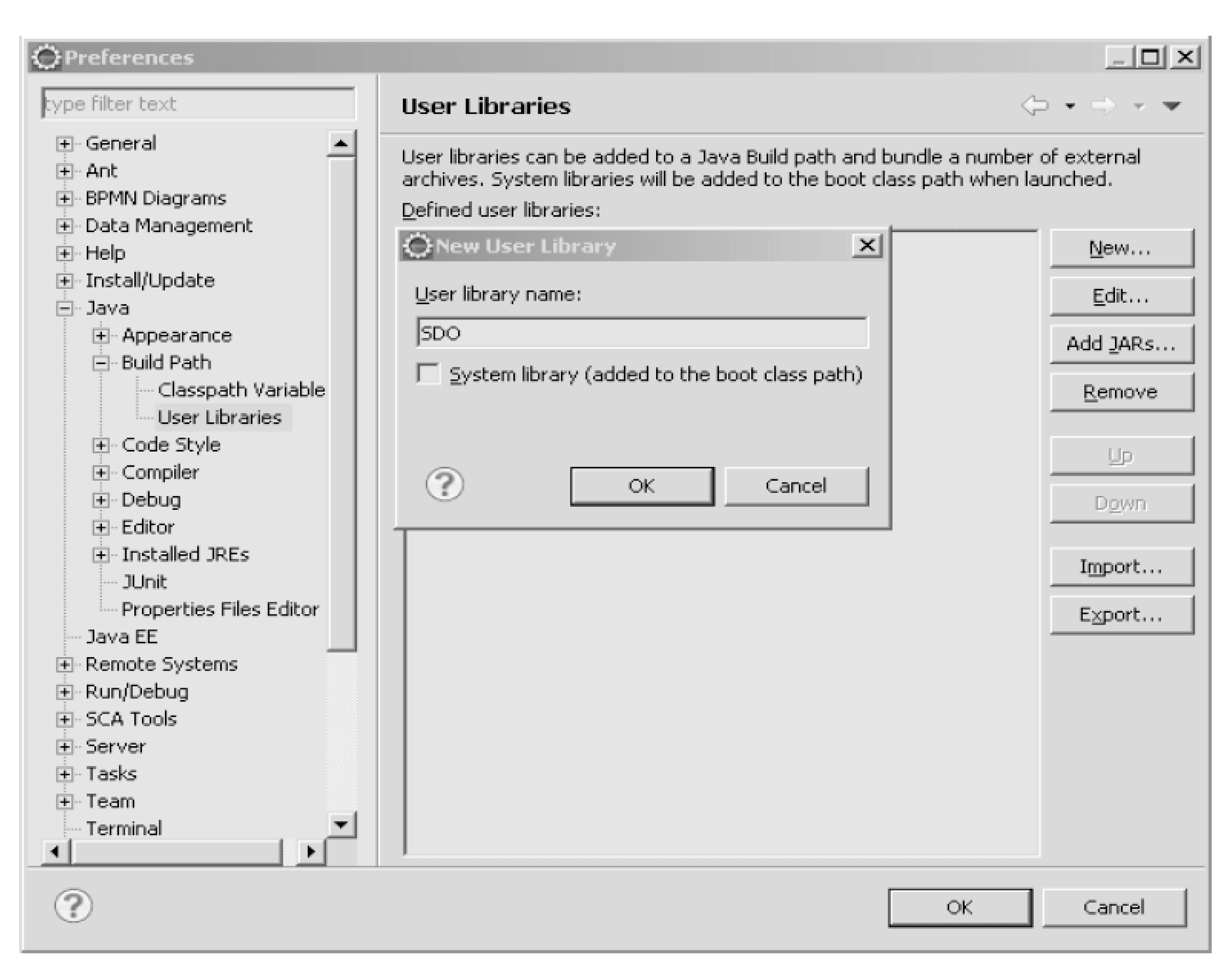


图 9.11　SDO 的配置

步骤 04　到此 SDO 环境就配置完成了，在实际应用中，需要将这个 user library 添加到工程的 build path 中。

9.3.2　Hadoop MapReduce 框架

Hadoop MapReduce 是 Google 提出的一个使用简易的软件框架，基于它写出来的应用程序能够运行在由上千台商用机器组成的大型集群上，并以一种可靠容错的方式并行处理上 T 级别的数据集。

MapReduce 是一个编程模型，也是一个处理和生成超大数据集的算法模型的相关实现。概念“Map（映射）”和“Reduce（化简）”，以及它们的主要思想，都是从函数式编程语言以及矢量编程语言借来的特性。采用 MapReduce 架构可以使那些没有并行计算和分布式处理系统开发经验的程序员有效利用分布式系统的丰富资源。

1. 分布式数据处理 MapReduce 简介

在过去的 5 年里，Google 的很多程序员为了处理海量的原始数据，已经实现了数以百计的、专用的计算方法。这些计算方法用来处理大量的原始数据，比如，文档抓取（类似网络爬虫的程序）、Web 请求日志等；也用来计算处理各种类型的衍生数据，比如倒排索引、Web 文档的图形结构的各种表示形式、每台主机上网络爬虫抓取的页面数量的汇总、每天被请求的最多的查询的集合等。大多数这样的数据处理运算在概念上很容易理解，然而由于输入的数据量巨大，因此要想在可接受的时间内完成运算，只有将这些计算分布在成百上千的主机上。如何处理并行计算？如何分发数据？如何处理错误？所有这些问题综合在一起，则需要大量的程序代码处理，因此使得原本简单的运算变得难以处理。

为了解决上述复杂的问题，Google 设计了一个新的抽象模型 MapReduce，使用这个抽象模型，程序员只要表述他想要执行的简单运算即可，而不必关心并行计算、容错、数据分布、负载均衡等复杂的细节。设计这个抽象模型的灵感来自 Lisp 和许多其他函数式语言的 Map 和 Reduce 的原语。

与传统的分布式程序设计相比，MapReduce 封装了并行处理、容错处理、本地化计算、负载均衡等细节，还提供了简单而强大的接口。通过这个接口，可以把大数据量的计算自动地并发和分布执行，使之变得非常容易。另外，MapReduce 也具有较好的通用性，大量不同的问题都可以简单地通过 MapReduce 来解决。

海量数据的运算大多数都包含这样的操作：在输入数据的“逻辑”记录上应用 Map 操作得出一个中间 key/value 集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，从而达到合并中间的数据，得到一个想要的结果的目的。使用 MapReduce 模型，再结合用户实现的 Map 和 Reduce 函数，就可以非常容地实现大规模并行化计算；通过 MapReduce 模型自带的“再次执行”（re-execution）功能，也提供了初级的容灾方案。

根据相关统计，用户使用 Google 搜索引擎仅仅进行一次关键字的查询，Google 的后台服务器都要进行数以千计的运算。如此庞大而复杂的数据处理量，没有非常好的并行处理和负载均衡，Google 服务不可能在用户的承受时间范围内返回结果，服务器负荷过重或者宕机，都会对用户的使用体验造成很恶劣的影响。而程序员使用 MapReduce 编程模型后，所有 Google 服务都保证了稳定快速的响应。在 Google 的集群上，每天都有 1000 多个 MapReduce 程序在执行。

2. 编程模型

MapReduce 以函数方式提供了 Map 和 Reduce 操作来进行分布式计算，利用一个输入 key/value 集合来产生一个输出的 key/value 集合。MapReduce 模型可以如下表示。

```
Map: (k1,v1) -> list(k2,v2)
Reduce: (k2,list(v2)) ->list(v3)
```

MapReduce 的运行模型如图 9.12 所示。

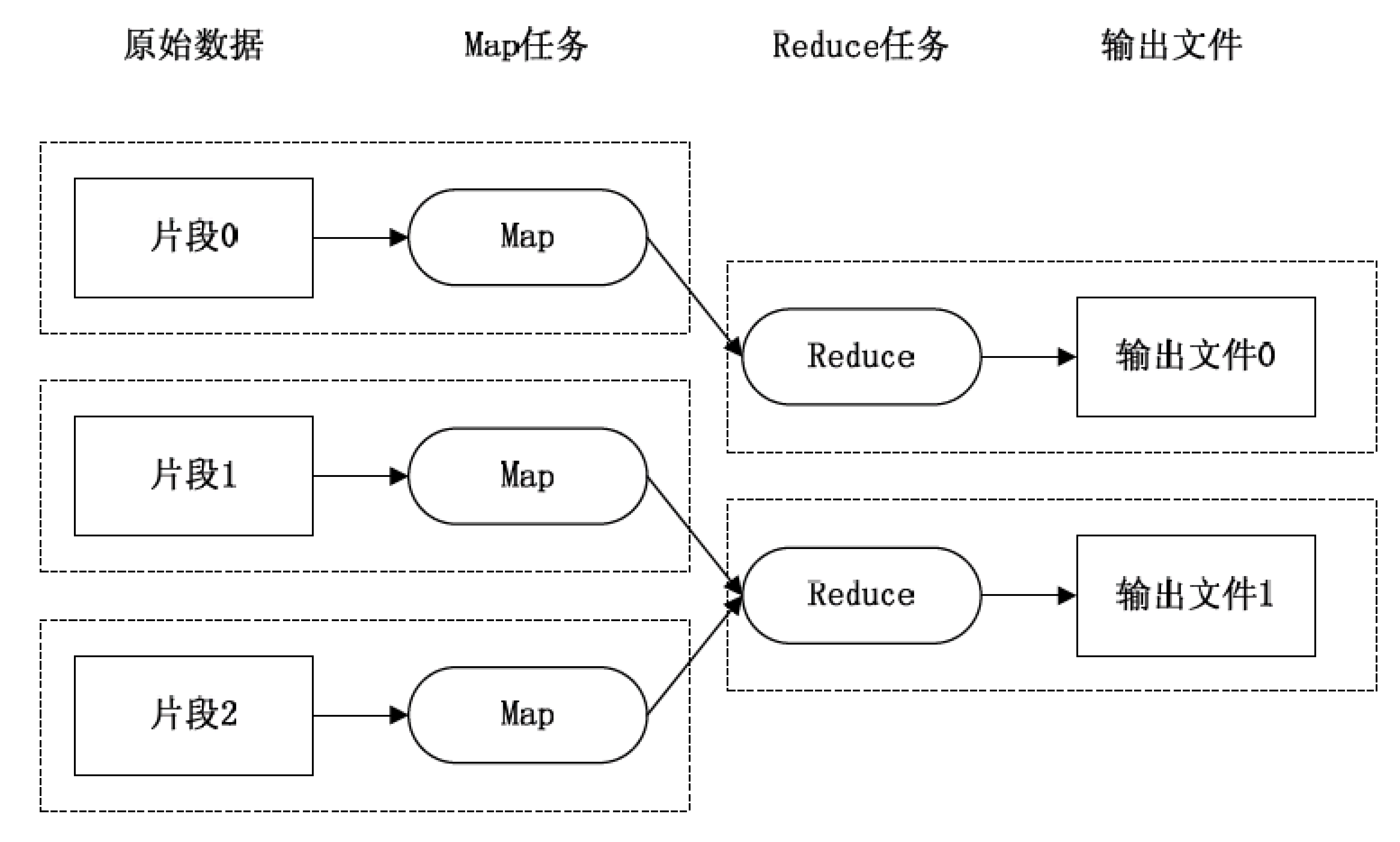


图 9.12　MapReduce 的运行模型

Map 任务是一类将输入记录转换为中间记录集的独立任务。用户自定义的 Map 函数接受一个输入的 key/value 值，然后产生一个中间 key/value 值的集合。输出 key/value 值的类型不需要与输入 key/value 值的类型一致。MapReduce 库把所有具有相同中间 key 值的 value 值集合在一起传递给 Reduce 函数。

Reduce 任务将一组与一个 key 关联的中间数值集规约（reduce）为一个更小的数据集。Map 任务的输出被排序后，就被划分给每个 Reduce 任务处理。用户自定义的 Reduce 函数接受一个中间 key 值和相关的 value 值的集合，然后合并这些 value 值，形成一个较小的 value 值的集合。一般地，每次 Reduce 函数调用只会产生 0 个或 1 个输出 value 值。通常通过一个迭代器把中间 value 值传递给 Reduce 函数，这样程序就可以处理无法全部放入内存中的大量的 value 值的集合。

9.3.3　HBase 数据库技术

HBase 即 Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。

HBase 是 Google Bigtable 的开源实现，类似于 Google Bigtable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。

图 9.13 中描述了 Hadoop System 中的各层系统，其中 HBase 位于结构化存储层，Hadoop HDFS 为 HBase 提供了高可靠性的底层存储支持，Hadoop MapReduce 为 HBase 提供了高性能的计算能力，Zookeeper 为 HBase 提供了稳定服务和 failover 机制。

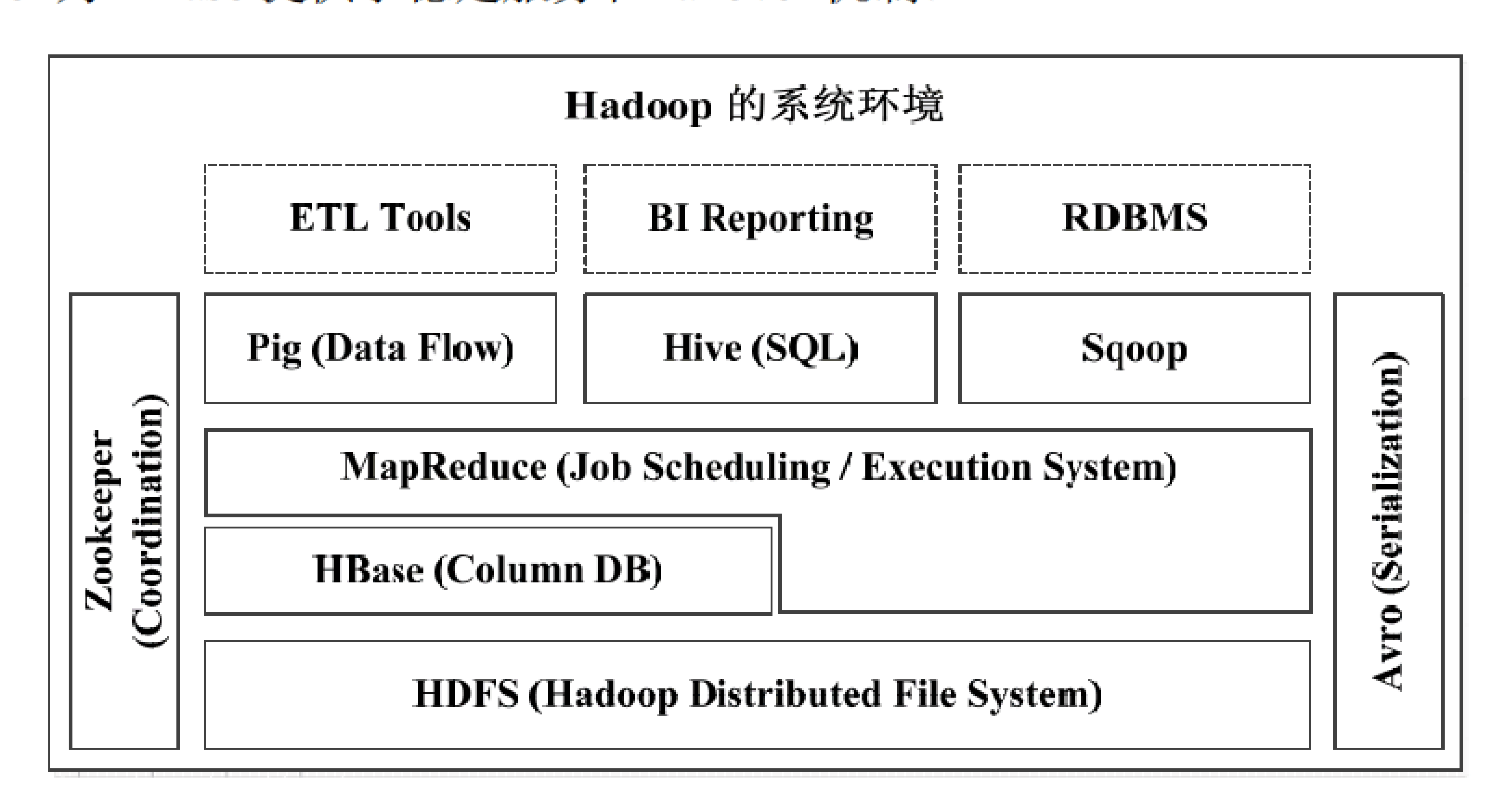


图 9.13　Hadoop 系统结构中的 HBase

此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变得非常简单。Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库数据向 HBase 中迁移变得非常方便。

下面列出了 HBase 访问接口。

- Native Java API，最常规和高效的访问方式，适合 Hadoop MapReduce Job 并行批处理 HBase 表数据。
- HBase Shell，HBase 的命令行工具，最简单的接口，适合 HBase 管理使用。
- Thrift Gateway，利用 Thrift 序列化技术，支持 C++、PHP、Python 等多种语言，适合其他异构系统在线访问 HBase 表数据。
- REST Gateway，支持 REST 风格的 HTTP API 访问 HBase，解除了语言限制。
- Pig，可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据，和 Hive 类似，本质最终也是编译成 MapReduce Job 来处理 HBase 表数据，适合做数据统计。
- Hive，当前 Hive 的 Release 版本尚没有加入对 HBase 的支持，但在下一个版本 Hive 0.7.0 中将会支持 HBase，可以使用类似 SQL 语言来访问 HBase。

HBase 系统架构如图 9.14 所示。

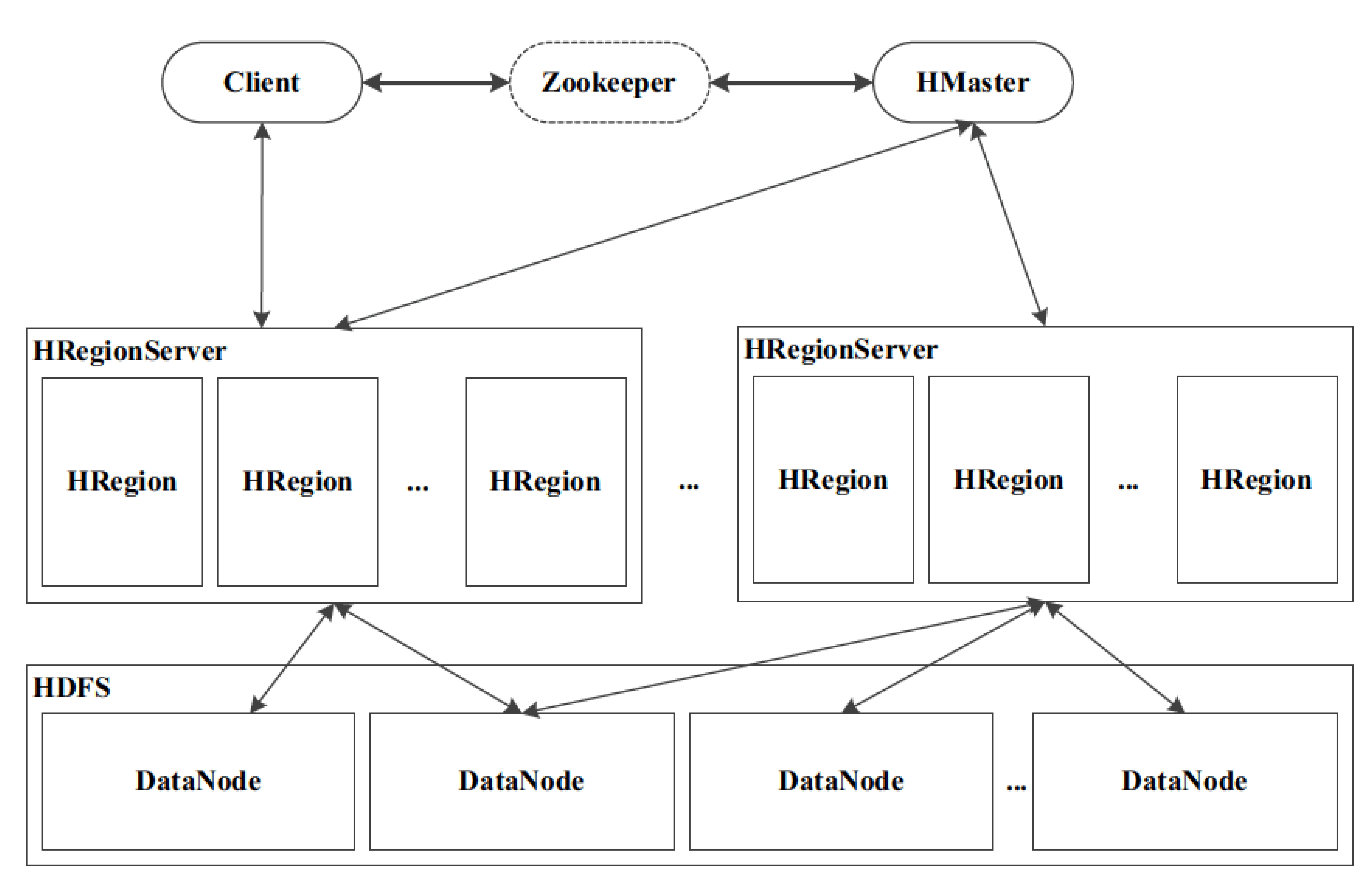


图 9.14　HBase 系统架构

（1）Client

HBase Client 使用 HBase 的 RPC 机制与 HMaster 和 HRegionServer 进行通信，对于管理类操作，Client 与 HMaster 进行 RPC；对于数据读写类操作，Client 与 HRegionServer 进行 RPC。

（2）Zookeeper

Zookeeper Quorum 中除了存储了-ROOT-表的地址和 HMaster 的地址，HRegionServer 也会把自己以 Ephemeral 方式注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个 HRegionServer 的健康状态。此外，Zookeeper 也避免了 HMaster 的单点问题。

（3）HMaster

HMaster 没有单点问题，HBase 中可以启动多个 HMaster，通过 Zookeeper 的 Master Election 机制保证总有一个 Master 运行，HMaster 在功能上主要负责 Table 和 Region 的管理工作：

- 管理用户对 Table 的增、删、改、查操作。

- 管理 HRegionServer 的负载均衡，调整 Region 分布。
- 在 Region Split 后，负责新 Region 的分配。
- 在 HRegionServer 停机后，负责失效 HRegionServer 上的 Regions 迁移。

（4）HRegionServer

HRegionServer 主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据，是 HBase 中最核心的模块。

9.3.4　模型驱动数据转换技术

模型驱动架构（Model Driven Architecture，MDA）是由对象管理组织（OMG）定义的一个软件开发框架。MDA 是一种基于 UML 以及其他工业标准的框架，支持软件设计和模型的可视化、存储和交换等。与 UML 相比，MDA 能够创建出机器可读和高度抽象的模型，且模型独立于具体的实现技术，以标准化的方式储存。

如图 9.15 所示，MDA 中将模型和元模型分为三层，M1 层是模型层，是开发人员直接面对的模型。M2 层称为元模型层，其中对应的是 M1 层模型的元模型，如 UML 和 SPEM 等。M2 层元模型中提取出不同领域的抽象概念和关系结构，为 M1 层的建模提供了建模符号。M2 层提供对应不同领域的领域建模语言。M3 层是元元模型层，也被称为 MOF 层。MOF 提供了定义 M2 层元模型所需要的更抽象一级的建模支持。MOF 是 M2 层中各个元模型的元模型，同时，MOF 也是自描述的，MOF 可以描述 MOF 元模型自身。在 MDA 框架中，M3 层只有 MOF 这一个模型，它是 MDA 中最基础和核心的标准，为 MDA 框架中的所有模型和/或元模型提供了统一的语义基础，MOF 解决了 M2 层中不同元模型之间的交互性，使得基于 MOF 的统一的模型操作成为可能。

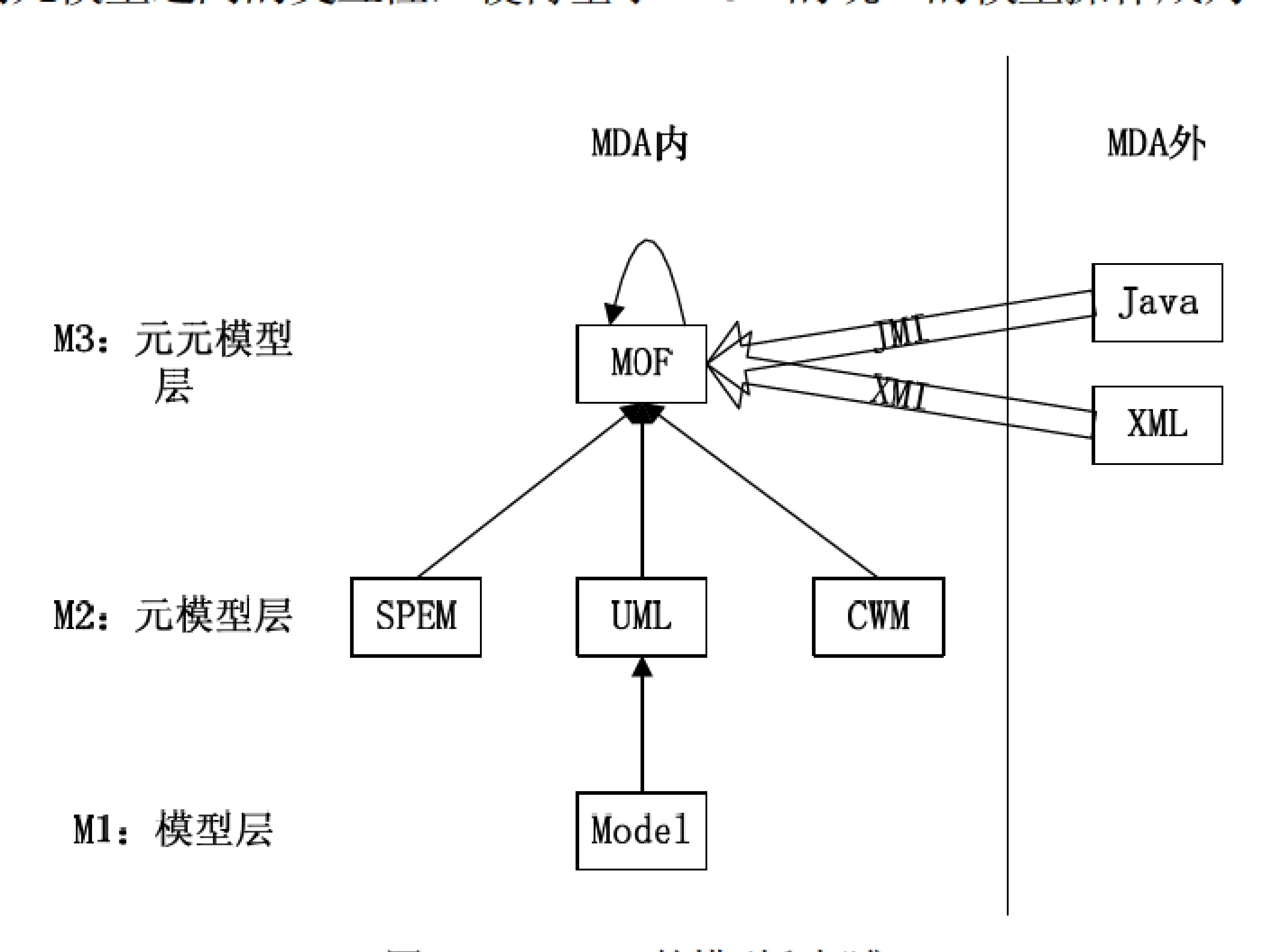


图 9.15　MDA 的模型和标准

其中重要的一点是 MOF 支持自省（introspection）机制，在 MOF 中定义了基于 MOF 的各级模型和元模型的统一的反射接口，如 RefBaseObject、RefObject、RefAssociation、RefPackage 等操作。

```
Reflective::RefBaseObject()  // 获取所有的 MOF 对象
Reflective::RefObject()      // 获取所有对象
Reflective::RefAssociation() // 获取所有 Association 对象
Reflective::RefPackage()     // 获取所有 Package 对象
```

通过上面介绍的接口，遵循 MOF 的程序实现不需要深入了解一个对象接口，即可以遍历各层的对象结构并找到需要的对象进行相关的操作，如创建、更新、访问和调用 M1 层对象实例的操作等。

在实际使用中，需要使用编程语言来实现这些接口并定义从 MOF 到（MDA 之外的）编程语言之间的映射，如定义 Java 中实现 RefObject()接口的规格，保证一个 Java 的 MOF 实现可以被其他用户统一地使用。如图 9. 15 右边部分所示，OMG 定义了从 MOF 到 Java、XML 等的映射。其中到 XML 的映射就是 XMI （XML Metadata Interchange）标准；到 Java 的映射就是 JMI（Java Metadata Interface）；正在制定中或即将制定的还有到 WSDL、.NET 的映射等。目前，XMI 已经广泛应用于各 UML 建模工具中，用于存储模型并在不同工具之间导入导出； JMI 也广泛应用各种基于 Java 的 MDA 工具中，如 AndroMDA 中使用 Sun 公司的 JMI 来实现 MDR。

除了图 9.15 中出现的 MOF、JMI 和 XMI 之外，MDA 中还有两个重要的标准，QVT 和 OCL。

- QVT（Query/View/Transformation）：模型转换标准，为基于 MOF 的元模型和/或模型之间的转换提供标准的转换规则描述语言。
- OCL（Object Constraint Language）：对象约束语言，用于配合 UML 和其他 M2 层元模型，精确地描述模型语义。

本章中将要用到的是 MDA 框架下的模型转换语言（ATL），ATL 是由 ATLAS 研究组开发出来的一种符合 OMG 的一个 QVT 提案的模型转换语言。ATL 属于基于规则的模型转换语言，其中使用了 OCL 作为约束描述语言。ATL 是一种混合语言，既有描述性语言的特征，又含有命令式语言的内容。作为一种基于规则的语言，描述性是其最主要特征，但是为了完成某些复杂的转换，命令式的内容也被加进去了。

首先介绍 ATL 的整体结构框架，如图 9. 16 所示为 ATL 的模型转换的层次结构。

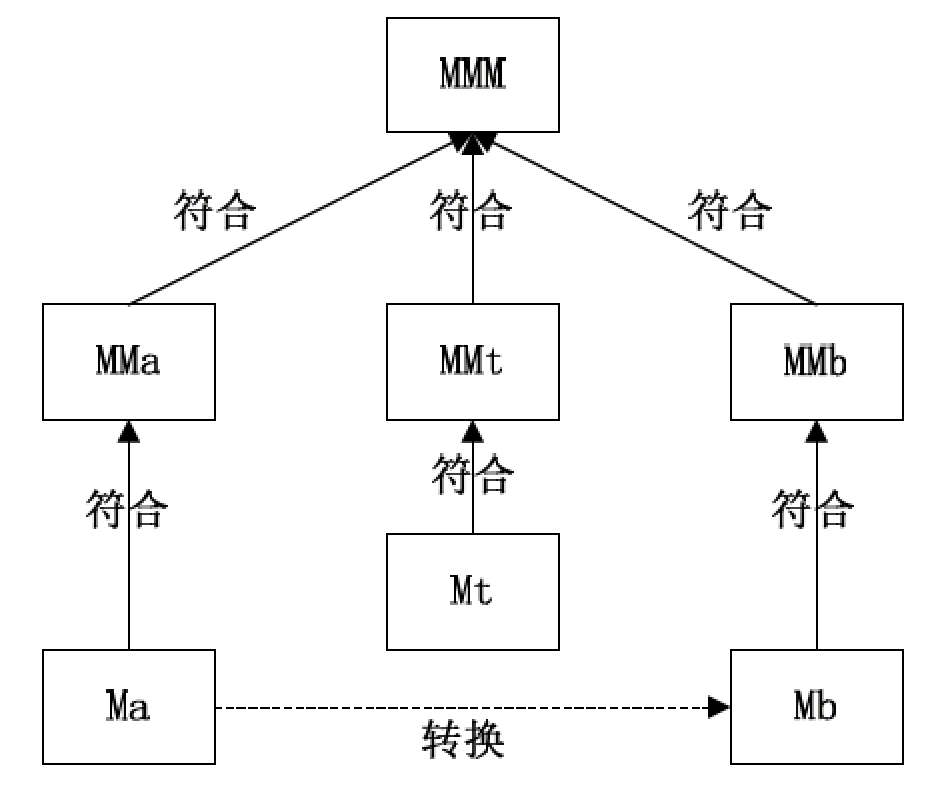


图 9.16　ATL 转换层次结构

图 9. 16 中 Ma 是源模型，而 Mb 是目标模型。Ma 符合其元模型 MMa，而 Mb 符合其元模型 MMb。同时 MMa 和 MMb 都符合唯一的元元模型 MMM。Mt 是一个模型转换实例，也是一种模型，Mt 符合模型转换的元模型 MMt。MMt 同时也符合唯一的元元模型 MMM。

在 ATL 中 Ecore 是其唯一的元元模型，其地位等同于 MOF。而 MMa 和 MMb 则是由 Ecore 创建出来的元模型。Ma 和 Mb 则是符合这些元模型的模型实例。MMt 已经被 ATL 定义好了，Mt 则是使用者自己要定义的模型转换模型，也就是模型转换程序或者被称为转换规则。

9.4 数据统一访问和灵活转换的详细设计与实现

数据统一访问和灵活转换的详细设计架构如图 9.17 所示。

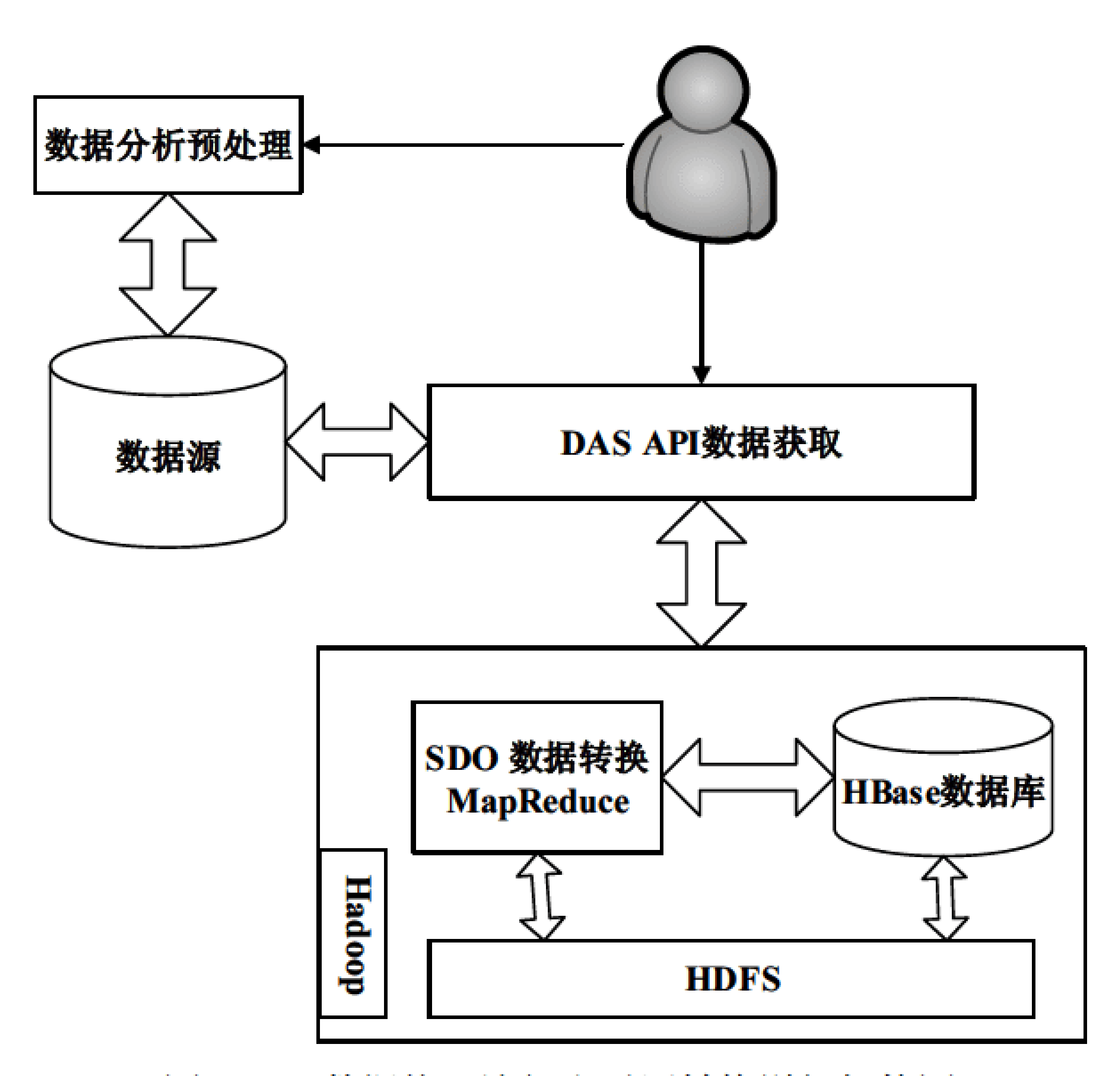


图 9.17　数据统一访问和灵活转换详细架构图

数据统一访问和灵活转换的设计思路是对数据源数据依据用户的要求作预处理分析，在预处理分析的基础上通过数据访问服务（DAS）对异构数据源实现统一访问，并采用基于服务数据对象（SDO）的技术结合 MapReduce 完成海量数据的并行转换应用处理。

9.4.1　数据分析及预处理

数据分析及预处理主要分为数据分析、数据审核与数据修正三部分。数据预处理的主要目的是保证数据能够满足大量数据的统一处理规则。数据分析与预处理流程如图 9.18 所示。

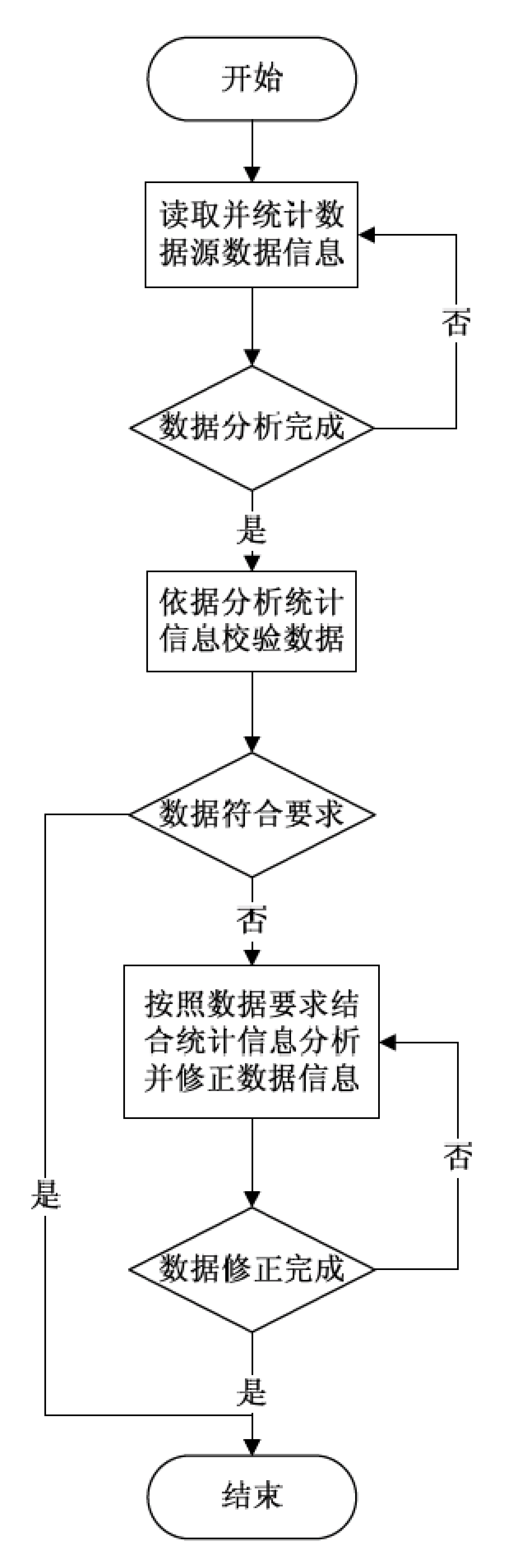


图 9.18　数据分析及预处理流程

1. 数据分析

数据分析（Data Profiling）是对数据可用性的一个统一检查和统计分析过程。在关系数据库中，数据库会对每张表做分析，一方面是为了让优化器可以选择合适的执行计划，另一方面，对于一些查询可以直接使用分析得到的统计信息返回结果，比如 COUNT（*）。Oracle 数据仓库构建工具 OWB 中提供的数据分析的统计信息更加全面，主要有记录数、最大值、最小值、最大长度、最小长度、唯一值个数、NULL 值个数、平均数和中位数等，针对字段的唯一值，在统计分析过程中也需要统一每个唯一值的信息，这对发现一些异常数据是非常有用的。

上述统计分析的统计信息可以联系统计学上面的统计描述，统计学上会使用一些统计量来描述一些数据集或者样本集的特征，在这里我们可以借助于类似 OWB 这样的 ETL 工具，也可以借助统计学的知识来分析，在统计学中有一个非常实用的图表工具——箱形图（Box plot），也叫箱线图、盒状图。可以尝试用箱形图来表现数据的分布特征，以便更直观地显示给用户数据的质量分布。一般的箱线图结构如图 9.19 所示。

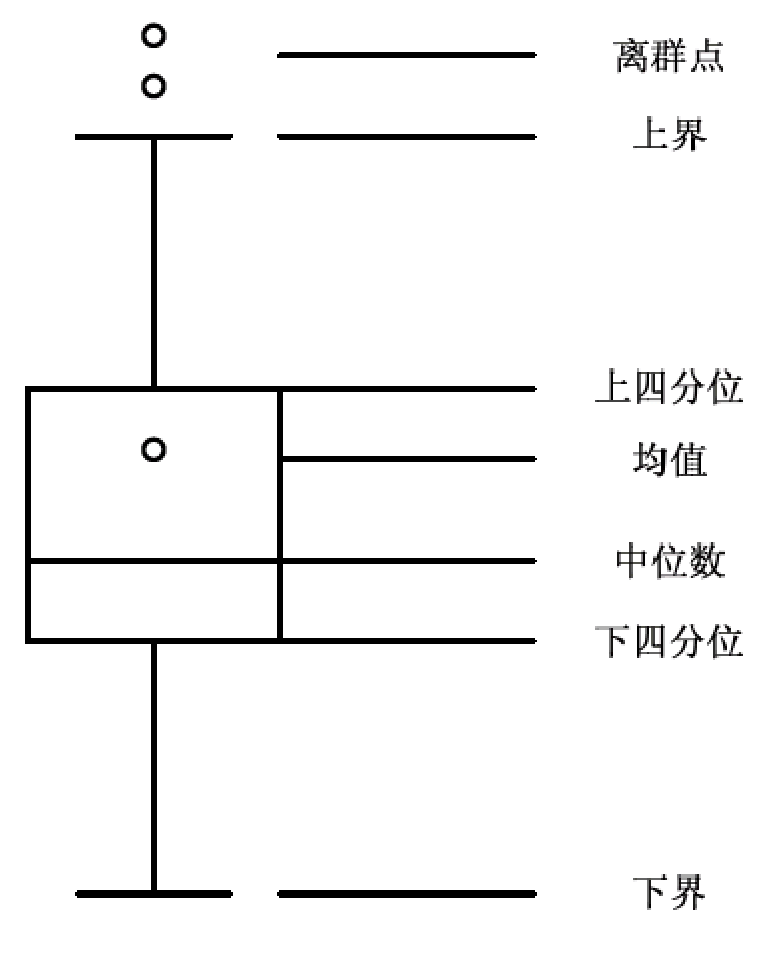


图 9.19　箱线图结构

箱线图有很多种表现形式，一般中间矩形箱的上下两边分别为数据集的上四分位数（占 75%比例：Q3）和下四分位数 （占 25%比例：Q1），中间的横线代表数据集的中位数（占 50%比例：Q2），箱线图中用“+”来表示数据集的均值。箱形的上下分别延伸出两条线，这两条线的末端，也称“触须”，一般是距离箱形 1.5 个 IQR（Q3-Q1，即箱形的长度），所以上端的触须应该是 Q3+1.5IQR，下端的触须是 Q1-1.5IQR；如果数据集的最小值大于 Q1-1.5IQR，我们就会使用最小值替换 Q1-1.5IQR 作为下方延伸线末端，同样如果最大值小于 Q3+1.5IQR，用最大值作为上方延伸线的末端，当最大或者最小值超出了 Q1-1.5IQR~Q3+1.5IQR 这个范围时，我们将这些超出的数据称为离群点或孤点，在图中表示出来就是图中在上方触须之外的点。当然，在一定的情况下也可以使用基于数据集的标准差 σ，选择上下 3σ 的范围，也可以使用置信水平为 95%的置信区间来确定上下边界的末端值。

箱线图并不能够展现数据集的全貌，但通过对数据集几个关键统计量的图形化表现，可以让我们看清数据的整体分布和离散情况。下面给出了关系数据库中数据统计信息的分析过程。

数据基本存储逻辑结构 E_R 设计如图 9.20 所示。

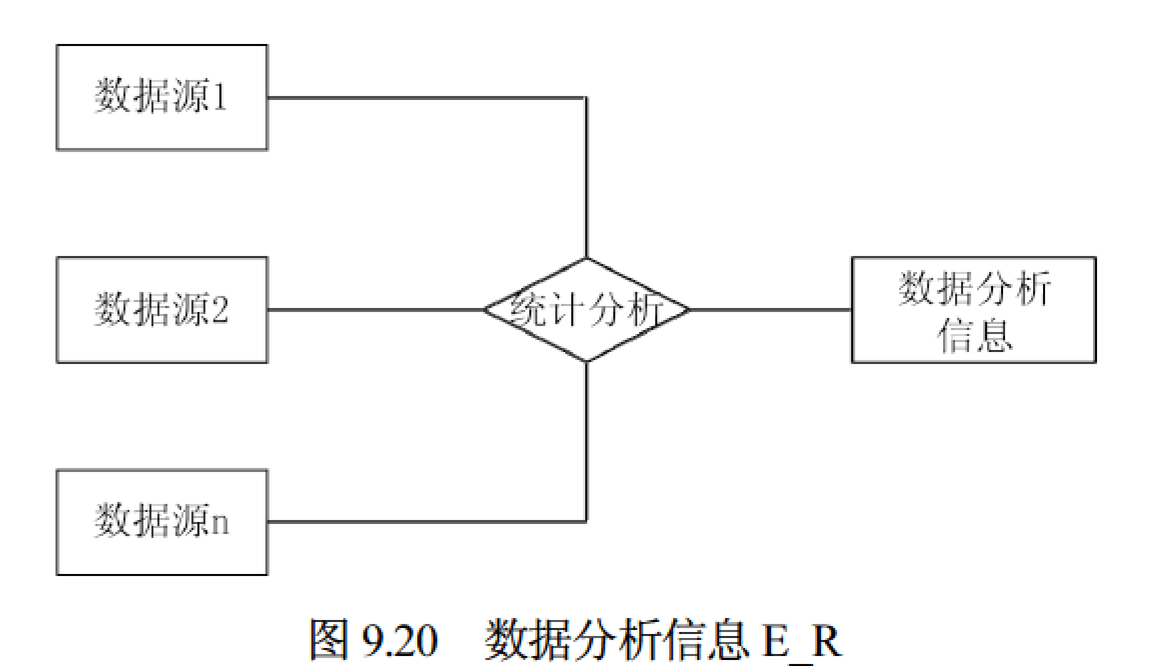


图 9.20　数据分析信息 E_R

依据 E_R 关系图设计关系数据库中的数据分析信息表结构如图 9.21 所示。

数据分析		
PK	标识	C-固定长度(10)
	数据源 字段名 最大值 最小值 记录数 最大长度 最小长度 唯一值个数 空值个数 平均数 中位数	C-可变长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10) C-固定长度(10)

图 9.21　数据分析信息表结构

这样将数据分析信息统一访问在一张表结构中，在关系数据库中往往能达到更高的性能。

ID 最大值统计分析如下：

```
SELECT MAX(ID) FROM TABLE
```

记录数统计分析语句如下：

```
SELECT COUNT(*)FROM TABLE
```

其他分析统计方式也类似。

通过数据分析可以得到数据的详细统计信息，接下来将对如何利用这些统计信息来审核数据的质量，发现数据中可能存在的异常和问题，然后对数据进行有效地修正，最后就可以得到符合统一访问访问等处理的“干净”数据。

2. 数据审核

基于数据分析中对数据源数据信息的统计分析信息，接下来可以对数据进行审核，以评估数据是否满足后期数据统一处理的规则的要求，其中数据的及时性主要跟数据的同步和处理过程的效率相关，更多的是通过监控新数据来保证数据的及时性，所以这里的数据审核主要指的是评估数据的完整性、一致性和准确性。

（1）完整性

从 Data Profiling 得到的数据统计信息里面看看哪些可以用来审核数据的完整性。首先是记录的完整性，一般使用统计的记录数和唯一值个数对比分析。比如网站每天的日志记录数是相对恒定的，大概在 1000 万上下波动，如果某天的日志记录数下降到了只有 100 万，那很有可能是记录缺失了。

完整性的另一方面是记录中某个字段的数据缺失，可以使用统计信息中的空值（NULL）的个数和数据记录数来分析进行审核。如果某个字段的信息理论上必然存在，这些字段的空值个数

的统计应该是零，在这类字段上可以使用非空（NOT NULL）约束保证数据的完整性；对于某些允许空的字段，空值的占比基本恒定，同样可以使用统计的空值个数来计算空值占比，如果空值的占比明显增大，很可能是字段的记录出现信息缺失。

（2）一致性

如果数据记录格式有标准的编码规则，那么对数据记录的一致性检验比较简单，只要验证所有的记录是否满足这个编码规则就可以，最简单的就是使用字段的长度、唯一值个数这些统计量。如果字段必须保证唯一，那么字段的唯一值个数和记录数必须一致。

一致性中逻辑规则的验证相对比较复杂，很多时候指标的统计逻辑的一致性需要底层数据质量的保证，同时也要有非常规范和标准的统计逻辑的定义，所有指标的计算规则必须保证一致。经常遇到的错误就是汇总数据和细分数据加起来的结果不一致，这种情况通常是数据在细分时有些无法明确归到某个细分项的数据给排除掉。审核这类数据逻辑的一致性，可以建立一些“有效性规则”，如 A>=B，C=B/A，那么 C 的值必须在 0~1 的范围内。当数据无法满足这些规则时就无法通过一致性检验。

（3）准确性

数据的准确性可能存在于个别记录，也可能存在于整个数据集。如果整个数据集的某个字段的数据存在错误，这种错误比较容易发现，利用数据分析的平均数和中位数也可以发现这类问题。当数据集中存在个别的异常值时，可以通过最大值和最小值的统计量去审核，或者使用箱线图的方法让异常记录直观易懂。对于字符乱码的问题或者字符被截断的问题，可以使用数据分布来发现这类问题，一般的数据记录基本符合正态分布或者类正态分布，那些所占比例异常小的数据项很可能存在问题。对于数值范围既定的数据，也可以进行有效性的限制，超过数据有效的值域定义数据记录就是错误的。

如果数据并没有显著异常，但仍然可能记录的值是错误的，只是这些值与正常的值比较接近而已，这类准确性检验比较困难，一般只能与其他来源或者统计结果进行比对来发现问题，如果使用多种数据收集系统或者数据分析工具，那么通过不同数据来源的数据对比可以发现一些数据记录的准确性问题。

通过数据审核可以发现数据中存在的一些问题，接下来主要针对数据审核中发现的问题对数据进行清洗和修正。

3. 数据修正

通过数据审核过程发现数据中存在的问题，这些问题一般可以采取一些方法进行修正，以便进一步提升数据的整体质量，一般可以通过以下几种方法进行修正。

（1）填补缺失值

对于记录缺失的问题，最简单的办法是数据回补。一般而言统计指标数据缺失可以从原始数据中重新统计获取，而原始数据缺失可以从抽取的数据源或者备份数据中回补。如果原始数据完全丢失，基本上就无法修正。

对于字段值的缺失，很多资料中都会介绍使用一些统计学中的方法进行修补，其实主要是对缺失数据值的预测或者估计，一般可以使用平均数、众数、前后数据值取平均等方法，或者使用回归分析的方法拟合指标的变化趋势后进行预测。这些方法在缺失数据值无法使用其他途径找回或者重新统计计算，并且在缺失数据值有变化规律可循的前提下都是可取的，当某天的指标值丢失时可以通过这类方法根据前几天的数据来预估该天的数值。一般的原则是，不太重要的字段如果缺失或者异常的记录数据所占比例小于 1%或者 5‰，可以直接过滤掉这些记录，如果所占比例比较高，需要进一步通过排查日志记录看是否存在问题。

（2）删除重复记录

数据集里面某些字段的数据值必须是唯一的，这些需要保证唯一的规则可以对数据库设置唯一约束，但在做数据处理时，为了能够保证数据加载过程中可以不因为违反唯一约束而中断，有时数据加载的过程需要较长的时间或处理成本，数据分析需要有容错能力以保证整个过程不被中断，而是先忽略重复记录，待整个数据分析过程完成后再对需要保证唯一的字段进行去重处理。

这些重复记录可以对数据分析过程中数据统计信息的唯一值个数和记录总数是否一致进行比较审核，而进行修正的最简单办法就是重复记录仅保留一条，删除其他记录。这需要根据现实情况，有时也可能使用把重复记录的统计量相加的方法进行去重。

（3）转化不一致记录

数据的转化是数据统一访问与转换数据过程中最常见的处理，因为数据统一访问与转换，需要把来自多个数据源的数据进行统一访问和转换应用，而不同数据源对某些含义相同的字段的编码规则会存在差异，即使是相同的数据源，也可能存在记录的不一致，如遇到新老版本的日志打到了一起，就会涉及数据的转化，但这种记录的不一致性会增大数据的处理成本。

在数据统一访问与转换处理时可能会遇到一些比较复杂的规则，这时最关键的还是对数据源记录方式的理解，较好理解数据源的记录方式才能最大程度地保证处理数据的一致。最好的处理方式就是能事先约定一套统一的数据记录和编码的方式，以降低后续的协调沟通和转化处理成本。

（4）处理异常数据

异常数据大部分情况是很难修正的，如字符编码等问题引起的乱码、字符被截断、异常的数值等，这些异常数据如果没有规律可循几乎不可能被还原，只能将其直接过滤。

有些数据异常则可以被还原，如原字符中掺杂了一些其他的无用字符，可以使用取子串的方法，或用 trim 函数去掉字符串前后的空格等；字符被截断的情况下如果可以使用截断后字符结合字典推导出原完整字符串，那么也可以被还原。数值记录中存在异常大或者异常小的值时，可以分析是否数值单位差异引起的，这类数值的异常可以通过数据转化进行处理，包括数值单位的差异都应该属于数据的不一致性，或者是某些数值被错误地放大或缩小。

9.4.2 基于 DAS 的数据源统一访问

异构数据源有各种形式，比如关系型数据库、文本、XML、Excel 等，而关系型数据库又有

MySQL、Oracle、SQL Server、DB2 等。数据源访问管理模块主要是屏蔽异构数据源的访问差异性，通过统一的接口访问。

数据访问结构如图 9.22 所示。

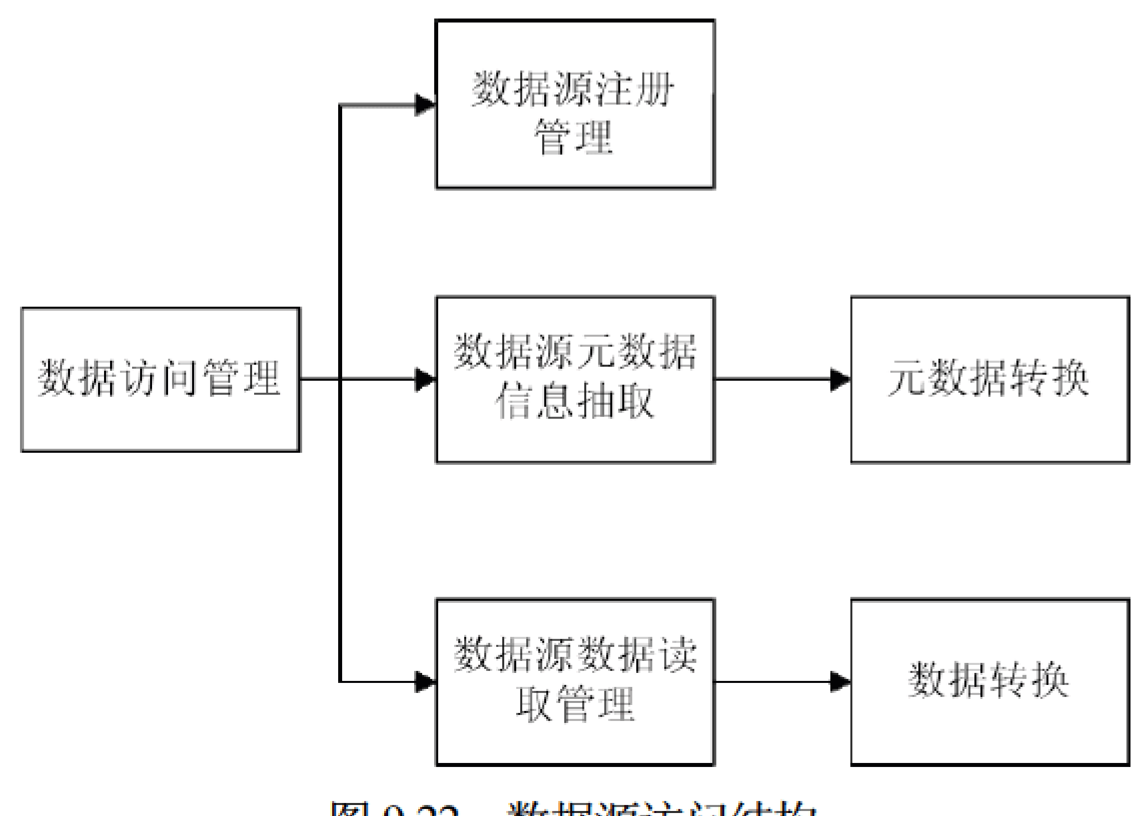


图 9.22　数据源访问结构

数据源访问模块主要由数据源注册管理、数据源元数据信息抽取和元数据转换以及数据源数据读取管理和数据转换组成。

1. 数据源注册管理

数据源注册管理信息使用 DataSource 实体进行存储。其中 sequenceNo、srvTitle、srvDesc、dbSystem、dbmsVersion 是数据源的相关基础信息的描述，而 dbDialect、dbDriver、conHostIP、dbAccessPort、dbName、dbUser、DBPin、dbSchema、dbProvider 是数据源访问权限相关的信息。dbDriver、conHostIP、dbAccessPort、dbUser、dbPin 分别代表数据源的驱动、物理地址、端口、用户名以及用户口令信息，是系统访问数据源的必备基本信息。DataSource 类结构如图 9.23 所示。

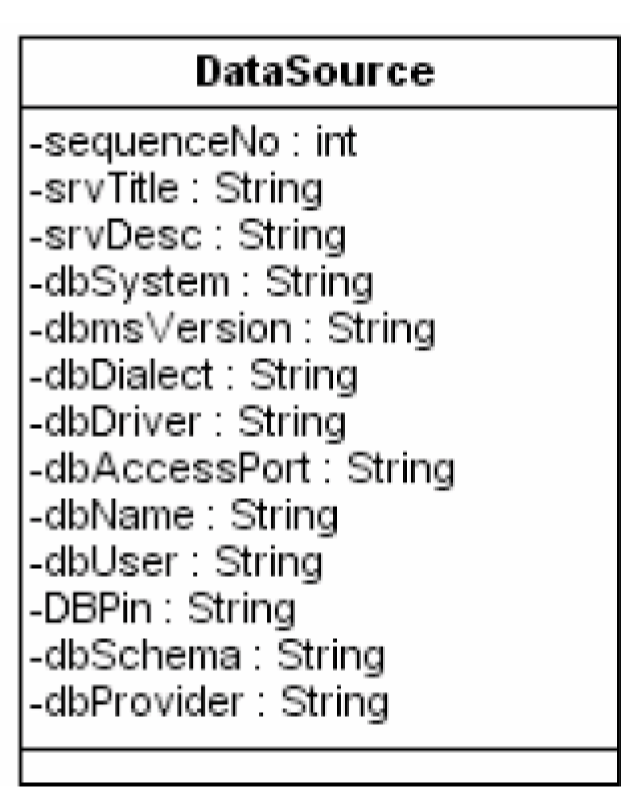


图 9.23　数据源注册 DataSource 类

其中 sequenceNo 作为标识数据源的 ID，做了唯一性约束。conHostIP 必须是 IP 地址格式的字符串，而 dbAccessPort 的字符串必须是 0~65535 之间的数字，因此进行了相关约束性定义。DataSource 实体为系统统一的数据集连接接口存储参数，是数据集连接实现“即插即用”的必备信息。

数据源管理类关系如图 9.24 所示。

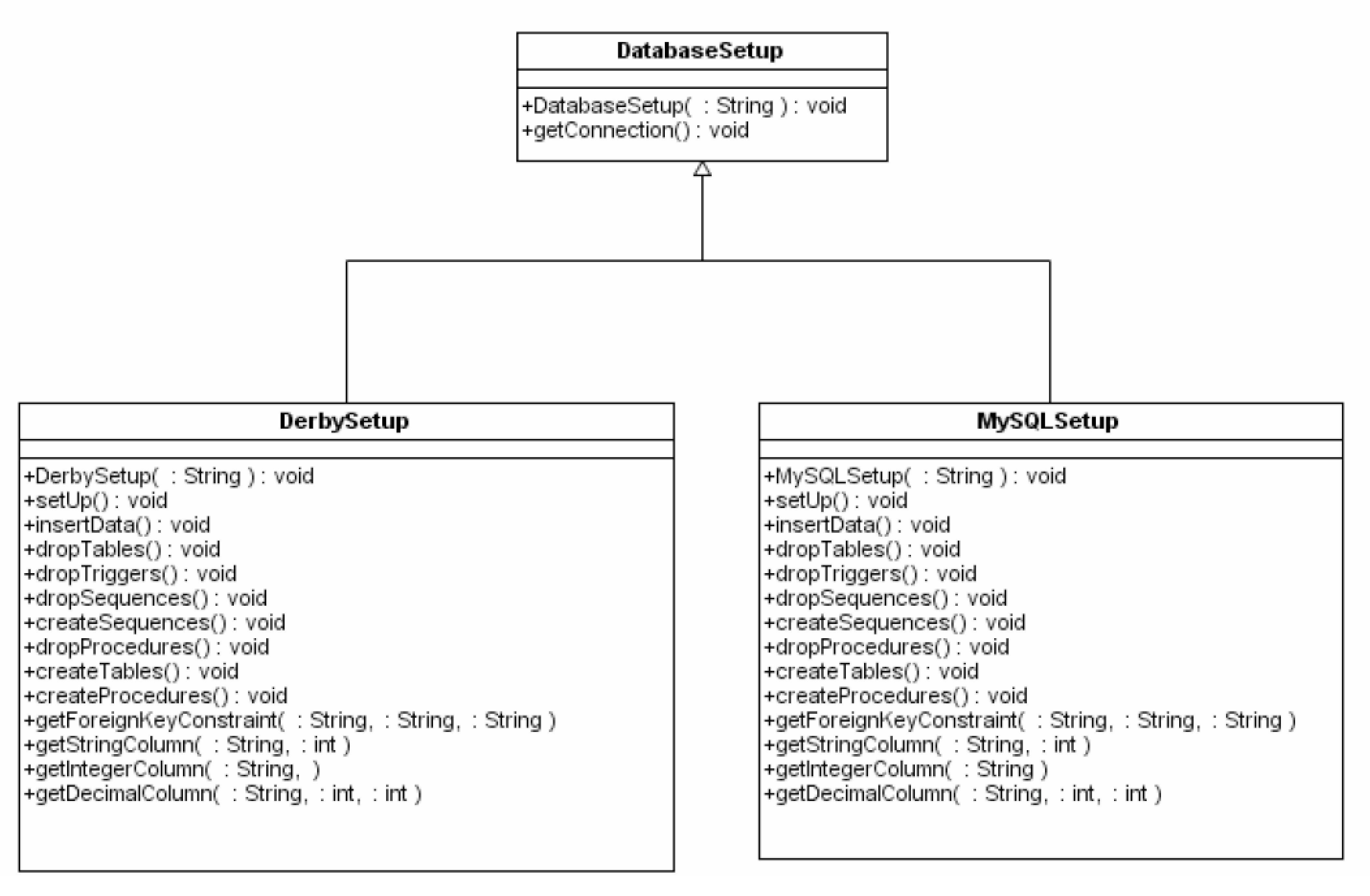


图 9.24　数据源管理类关系

图 9.24 中 DatabaseSetup 类为数据源配置管理抽象类，其各个子类分别实现了对具体数据源的配置访问管理。DerbySetup 子管理类实现了对 Derby 数据源的访问管理，MySQL Setup 子管理类实现对 MySQL 数据库的操作管理。

2. DAS 访问配置管理

基于 DAS 的数据访问服务对 Derby 数据源的访问操作管理配置文件的配置过程实例如下：

```
<DBConfig xmlns="http:///org.apache.tuscany.das.rdb/dbconfig.xsd">
 <ConnectionInfo>
     <ConnectionProperties
         driverClass="org.apache.derby.jdbc.EmbeddedDriver"
         databaseURL="jdbc:derby:target/dastest; create = true"
         loginTimeout="600000"/>
 </ConnectionInfo>
 <Table name="COMPANY" SQLCreate="CREATE TABLE COMPANY (ID INT PRIMARY KEY NOT NULL,
NAME VARCHAR(30), EOTMID INTEGER)">
     <row>51, 'ACME Publishing', 0</row>
     <row>52, 'Do-rite plumbing', 0</row>
     <row>53, 'MegaCorp', 0</row>
```

```
    </Table>
    <Table name="DEPARTMENT" SQLCreate="CREATE TABLE DEPARTMENT (ID INT PRIMARY KEY
NOT NULL GENERATED ALWAYS AS IDENTITY, NAME VARCHAR(30),LOCATION VARCHAR(30),
DEPNUMBER VARCHAR(10),COMPANYID INT)">
        <row>'Advanced Technologies', 'NY', '123', 1</row>
        <row>'Default Name', '', '', 51</row>
        <row>'Default Name', '', '', 51</row>
        <row>'Default Name', '', '', 51</row>
        <row>'Default Name', '', '', 51</row>
        <row>'Default Name', '', '', 51</row>
        <row>'Default Name', '', '', 51</row>
    </Table>
    <Table name="EMPLOYEE" SQLCreate="CREATE TABLE EMPLOYEE (ID INT PRIMARY KEY NOT
NULL GENERATED ALWAYS AS IDENTITY,NAME VARCHAR(30),SN VARCHAR(10), MANAGER SMALLINT,
DEPARTMENTID INT)">
        <row>'John Jones','E0001',0,12</row>
        <row>'Mary Smith','E0002',1,null</row>
        <row>'Jane Doe','E0003',0,12</row>
        <row>'Al Smith','E0004',1,12</row>
    </Table>
   </DBConfig>
```

数据源配置管理 CustomerDatabaseInitializer 的实现如下：

```
   public CustomerDatabaseInitializer(String configFile) {
      this.config = ConfigUtil.loadConfig(this.getClass().getClassLoader().
getResourceAsStream(configFile));
   if(config.getConnectionInfo().getConnectionProperties().getDriverClass().index
Of(DERBY != -1) {
   dbType = DERBY;
   }
   else if(config.getConnectionInfo().getConnectionProperties().getDriverClass().
   indexOf(MYSQL)!= -1)  {
            dbType = MYSQL;
         } else {
            dbType = null;
         }
         //get connection info from config
        dbURL = config.getConnectionInfo().getConnectionProperties().
getDatabaseURL();
```

```
        user = config.getConnectionInfo().getConnectionProperties().
getUserName();
        password = config.getConnectionInfo().getConnectionProperties().
getPassword();
    }
```

数据源配置初始化实现如下：

```
    public void Initialize() {

        //display DB configuration iformation
        DisplayDatabaseConfiguration();

        //initialize DB
        try {
            if(dbType.equals(DERBY)){
                new DerbySetup(dbURL+"-"+user+"-"+password);
            }

            If(dbType.equals(MYSQL)){
                new MySQLSetup(dbURL+"-"+user+"-"+password);
            }

        } catch(Exception e){
            throw new RuntimeException("Error initializing database !", e);
        }
    }
```

3. 数据源元数据抽取模块

该模块在于获取各异构数据源的元数据信息。对于不同类型的数据源，获取的内容是不同的。比如，对于文本类型的数据源，需要获取文本路径、抽取起始行、字段分隔符，并需要定义源表名以及各字段的元数据信息（名称、类型、长度、精度）；对于关系型数据库类型的数据源，需要获取数据库类型、数据库所在的 IP 地址、数据库名、账号、密码以及各字段的元数据信息。这些信息通过元数据管理模块来保存在元数据资料库中。

数据源基本信息结构定义及元数据抽取实现如下。

属性元数据信息结构定义：

```
    Class Attr
    {
     String  AttrName;
```

```
 String  AttrType;
 Integer AttrMaxLength;
 String  AttrDefault;
 Boolean PK;
 Boolean FK;
 Boolean Union;
}
```

表内容元数据信息结构定义：

```
Class Table
{
 String  TabName;
 Boolean HasPK;
 Arraylist< Attr > TabAttr;
}

ReadDataSourceMetaData(DataSource ds)
{
     TYPE=ds.DBTYPE;      //数据源类型
     DBIP=ds.DBIP;        //数据源 IP
     DBPort=ds.DBPort;    //数据源端口
     DBName=ds.DBName;    //数据源名
     DBUser=ds.DBUser;    //数据源用户
     DBPwd=ds.DBPwd;      //数据源密码
     List<Table> TableMetaDataList =ds.ReadTableInfo();
}

List<Table> ReadTableInfo()
{
 List<Table> tableData;
 For tableIndex in tablelist
 {
     Table=ReadaTable();
     TableData.Add(Table);
 }
}
```

4. 元数据转换模块

元数据的典型表现为对对象的描述，即对数据库、表、列、列属性（类型、格式、约束等）以及主键 / 外键关联等的描述，特别是现行应用的异构性与分布性越来越普遍的情况下，统一的元数据就愈发重要了。在一个数据源里，数据类型是非常多的，仅 Oracle 的字符类型就有 CHAR、

VARCHAR2、NCHAR、NVARCHAR2 等，而数据源的多样性使数据类型的多样性更加明显。在异构数据转换过程中，同为数值型，可能因为精度不匹配而报错，同为字符型，可能因为来自不同的数据源而无法联合操作，类似的问题非常多。针对这些问题，采取了如下策略：收集分析各数据源所支持数据类型，建立一个数据类型库，而对于文本文件的数据一律看作字符型数据。

数据处理中心所支持的三种标准数据类型有：string、numeric 和 date。

- string 代表数据类型源中的各种字符类型。
- numeric 代表数据类型源中的各种数值类型。
- date 代表数据类型源中的各种时间和日期类型。

元数据映射规则表如表 9.9 所示。

表 9.9　元数据映射规则

数据类型	所映射类型
CHAR	String
VARCHAR2	
NCHAR	
NVARCHAR2	
Int	Numeric（char[n，m]）
Double	
Float	
TimeStamp	Date（char[n]）
Date	
...	...

5. 数据源数据读取

该模块完成实际数据的读取功能，包括将数据源的数据读取到数据处理中心。各数据源的数据在存储结构上存在较大的差异性，在数据源数据读取过程中涉及海量的数据，动辄百万行，对数据处理的性能提出了巨大挑战，尤其是从数据源端读取数据阶段，大量的 IO 操作可能成为数据读写过程中的性能颈瓶，可以通过索引、使用 Bulk Loading 和 External Loading、建立连接池和缓冲池等方式来提高效率，利用各个数据库的一些独特的优化技巧，来提高数据的读取速度。通过 DAS 将数据源数据加载到数据对象中后对数据对象的处理都可以采用标准的 SDO 方式，因此这里采用基于 SDO 的技术统一来处理数据，以更大程度地提高数据处理性能。

数据源数据获取数据流图如图 9.25 所示。



图 9.25　数据源数据获取数据流图

图 9.25 中描述了数据源数据访问过程的数据流导向。

基于 SDO DAS 的数据统一访问类关系如图 9.26 所示。

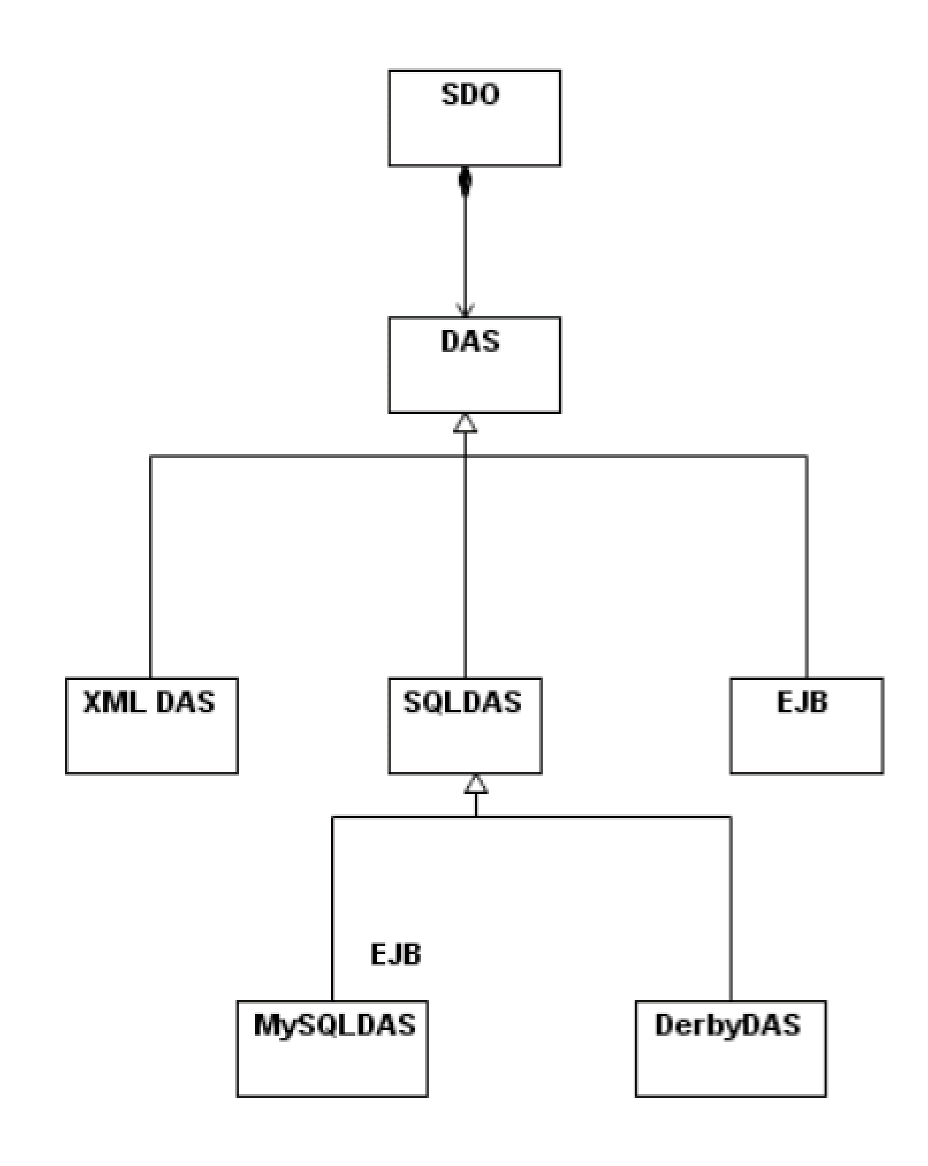


图 9.26　数据统一访问类关系

图 9.26 中详细描述了各相关类之间的关系。

接下来通过 SDO DAS 访问 XML 数据源中的数据并实现数据的处理等操作。SDO 数据访问前需要用户注册数据类型，数据类型的注册有通过 XSD 文件格式注册或用户在程序中动态注册两种方式。在下面的介绍中将以 XSD 文件格式注册并在此基础上实现 XML 数据的统一访问。

用 XSD 文件定义数据对象的类型描述：

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:sdo="commonj.sdo"
    xmlns:sdoxml="commonj.sdo/xml"
    xmlns:company="mynamespace"
    targetNamespace="mynamespace">
    <xsd:element name="company" type="company:CompanyType"/>
     <xsd:complexType name="CompanyType">
         <xsd:sequence>
             <xsd:element name="Departments" type="company:DepartmentType"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="Director" type="company:DirectorType"/>
            <xsd:attribute name="UnionRep" type="xsd:IDREF" doxml:propertyType=
```

```
"company:EmployeeType"/>
        </xsd:complexType>
            <xsd:complexType name="PersonType">
            <xsd:attribute name="Name" type="xsd:string"/>
        </xsd:complexType>
        <xsd:complexType name="DepartmentType">
            <xsd:restriction base="company:PersonType" />
            <xsd:sequence>
                <xsd:element name="Employees" type="company:EmployeeType"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="EmployeeType">
            <xsd:restriction base="company:PersonType" />
            <xsd:attribute name="Serial Number" type="xsd:ID"/>
            <xsd:attribute name="Desk Location" type="xsd:string"/>
        </xsd:complexType>
       </xsd:schema>
```

通过使用 XML 数据访问服务类 XSDHelper 加载上面定义的数据源数据类型文件（XSD 文件）。

```
    XSDHelper.INSTANCE.define(ClassLoader.getSystemResourceAsStream(PO_XSD_RESOURC
E), null);
```

以下面的 XML 数据作为数据源：

```
    <?xml version="1.0" encoding="UTF-8" ?>
    <company xmlns="mynamespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"name="ACME" >
        <Departments Name="Sales"    >
            <Employees>
             <Name>Helena B Carter</Name>
       <\Employees>
       <Employees>
             <Name>Anthony W Thompson</Name>
       </Employees>
    </Departments>
    </company>
```

通过使用 XML 数据访问服务类 XMLHelper 从 XML 数据文件中读取数据，并通过

getRootObject()方法返回数据对象（Data Object）。

```
    company= XMLHelper.INSTANCE.load(ClassLoader.getSystemResourceAsStream(PO_XML_
RESOURCE)).getRootObject();
```

接着创建数据图（Data Graph），并将 company 数据对象设置到数据图中。SDOUtil 是 Tuscany 中最重要的 SDO 工具类之一，主要用来创建数据对象、数据图等重要的操作。

```
    DataGraph dataGraph = SDOUtil.createDataGraph();
    SDOUtil.setRootObject(dataGraph, company);
```

从数据图中获取变更摘要，并开启对数据图进行变更的记录，比如修改数据对象 company 中 Departments 属性值的过程如下：

```
    ChangeSummary changeSummary = dataGraph.getChangeSummary();
    changeSummary.beginLogging();
    company.setString(company.getProperty("Departments"),"Depart");
    changeSummary.endLogging();
```

可以使用变更摘要的 isModifed 方法返回数据图是否发生更改，如果发生了更改可以使用 getOldValue 方法获取更改前的值，也可以使用 undoChanges 方法回滚数据图。其操作方式简单统一，降低了对复杂数据的统一处理难度。

9.4.3　映射模式表示与数据存储管理模块

1. 映射模式管理

映射模式主要用来管理异构数据源间元数据的映射。建立数据源类型到标准类型映射关系表，保存在元数据库中，当把数据源的数据读取到数据处理中心时，数据类型统一转化为标准数据类型，以此屏蔽数据源的差异，以及往目标系统存储数据时，参照类型映射关系表完成字段类型的识别、匹配。

源数据项的表示定位到具体的数据源数据后，若源数据项不唯一，那么源数据并不能够直接映射到目标数据，因为源数据间存在关系，如相加的关系、列变行、行变列等，需要表示这种关系。

将数据源与数据源间存在的关系归为以下几类。

（1）二元运算关系

二元运算关系是针对两个数据项而言的，即数据项有且仅有两项。两个数据项之间的关系，最常见的是四则运算关系，即加、减、乘、除。除此之外，还有求百分比等。

（2）多元运算关系

多元运算关系指的是数据项在两个或者两个以上的数据关系。常见的有选择多个数据的最大

值、最小值、平均值、偏差等。

多元运算关系与二元运算关系最大的不同是：多元运算关系中数据项的数目是不确定的。数据项数目改变并不改变数据的处理方式。

（3）行列运算关系

行列运算关系是属于多元运算关系的特殊情况，由于它本身存在特定的规律，在此专门归为一类。常见的有行变列、列变行、多列相加等。

行列运算可以分为两类：

- 单个行列运算关系

最常见的情况是行变列、列变行。

- 多个行列运算

常见的有行相加、列相加等。

（4）自定义运算关系

自定义运算关系是针对用户具体的业务需求而言的。这种运算关系非常广泛，随着用户不同，定义的运算关系也不同。

数据源与数据源之间的关系表示规则的 BNF 范式定义如下所示：

```
<异构数据源间的关系>::=
<源数据项><二元运算符><源数据项>
|<多元运算符>"("<源数据项>{,<源数据项>}")"
|<行列运算符>"(" <源数据项列或行>{,<源数据项的列或行>}")"
|<自定义运算符>"("<源数据项>{,<源数据项>}")"
```

定义了目标数据项、源数据项、源数据项间关系的表示规则后，就能够以表达式的形式来表示源、目标数据的映射关系。

数据映射关系规则 BNF 表示如下所示：

```
<源、目标数据映射关系>::=
          <目标数据项>
          "="<单个异构数据项>
          |<多个异构数据项间的关系>
```

源数据到目标数据的映射关系表达式是按照一定步骤确立的，具体确立的步骤如下：

步骤01 根据目标表的设计，表示目标表的各数据项。在表示数据项时，可能会表示各单个数据项，也可能表示数据行或列。

步骤02 目标数据项的表示，定义与之相关的源数据表示。

步骤03 选择异构数据间的关系。若存在多个异构数据，则需要确立多个数据之间的关系。

步骤04 定义并存储源数据到目标数据的映射表达式。

步骤05　编译或解析表达式。

2. 基于 HBase 的数据存储

前面讲述了元数据的重要性并描述了元数据的抽取转换过程等。接下来主要讲述基于 HBase 的数据转换规则等数据以及其他一些中介数据存储。在数据转换执行时，通过读取映射关系元数据而得到数据转换的脚本或存储过程。它是数据应用和数据转换之间的重要纽带。根据数据存储需求，对数据库的设计方面需要能够满足元数据的存储以及映射模式的快速查找等特性。在确定数据库的总体设计思路后，进一步完成对系统中数据库表的设计。

下面给出基于 HBase 的数据库存储结构设计。

数据源基本元信息表的基本结构设计如表 9. 10 所示。

表 9.10　数据源基本元信息表

表名		DBInfo
行关键字 Row Key		数据源唯一序列号 ID （或数据源 IP：端口号）
列族 Column family：		DBMetaData_Info
列	DBMetaData_Info：：	IP
	DBMetaData_Info：：	Port
	DBMetaData_Info：：	DBName
	DBMetaData_Info：：	DBUser
	DBMetaData_Info：：	DBPwd

数据源详细元信息结构存储表设计如表 9.11 所示。

表 9.11　数据源详细元信息结构存储表

表名		TableInfo
行关键字 Row Key		数据源唯一序列号 ID （或数据源 IP：端口号：：表名）
列族 Column family：		TableMetaData_Info
列	TableMetaData_Info：：	TabName
	TableMetaData_Info：：	AttrName
	TableMetaData_Info：：	AttrType
	TableMetaData_Info：：	AttrMaxLength
	TableMetaData_Info：：	PK
	TableMetaData_Info：：	FK
	TableMetaData_Info：：	Union

数据源异构数据映射关系存储表结构设计如表 9.12 所示。

表 9.12　数据源异构数据映射关系存储表

表名		DBMapRel
行关键字 Row Key		数据源唯一序列号 ID （或数据源 IP：端口号：：RELID）
列族 Column family：		DBMap_Info
列	DBMap_Info：：	SRC
	DBMap_Info：：	DEST
	DBMap_Info：：	MapFunction

用户定义好映射关系表达式，数据处理中心将表达式存储在 DBMapRel 表中的 MapFunction 列下，在 DBMapRel 表中，这些记录可能是目标数据行或列的映射，也可能是目标数据项的映射。在基于 HBase 的数据库设计中通过过滤器可以很快获得单个所需的数据映射关系表达式或元数据信息等，同时 HBase 数据库也支持自定义 Filter 来扩展数据库默认的过滤器，进而可以满足用户自定义的查询规则，以实现更具体的或更抽象的查询方式。

HBase 数据库管理类图如图 9.27 所示。

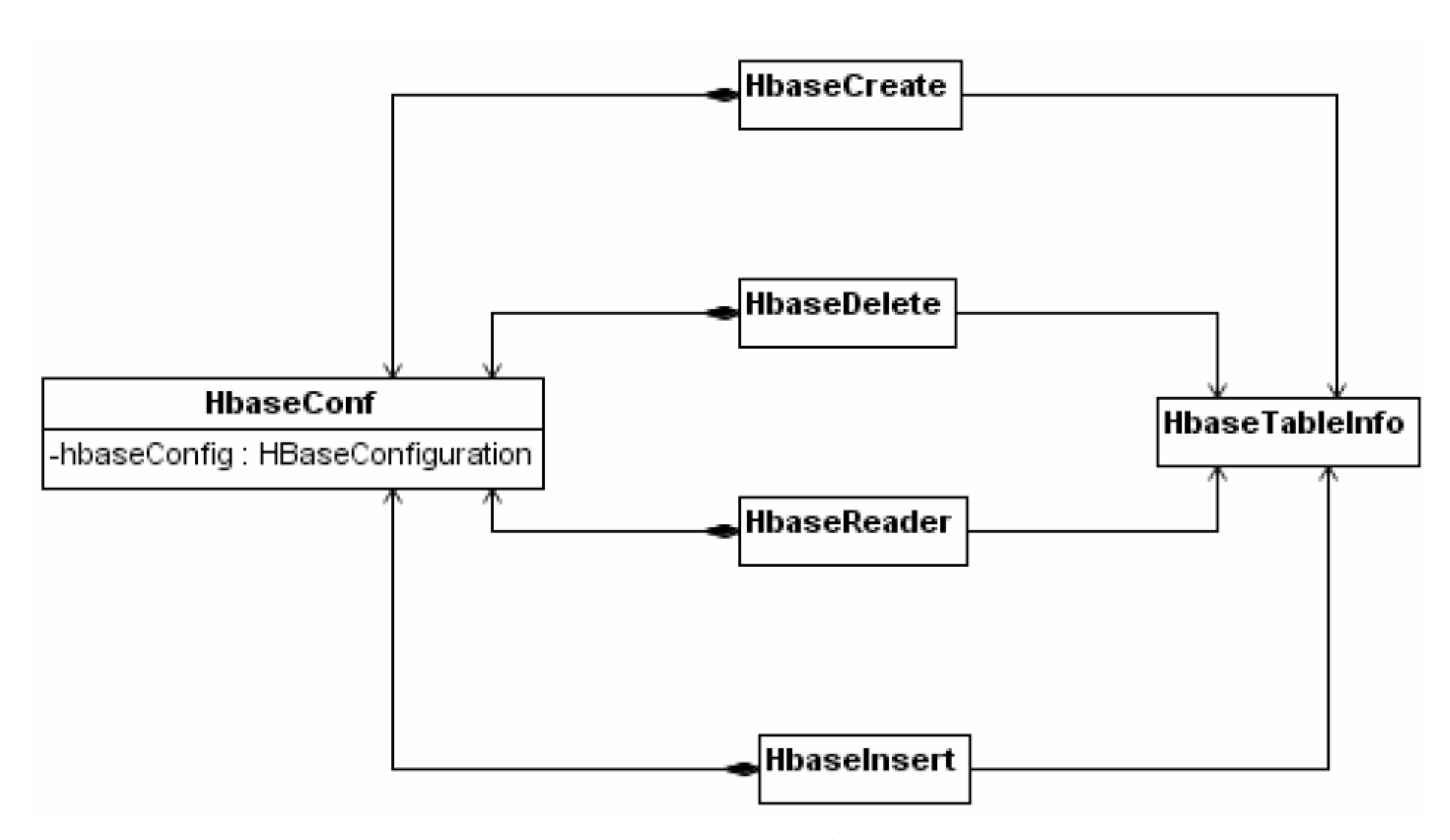


图 9.27　HBase 数据库管理类图

HBase 数据存储管理相关类实现如下。

```
//HBase 配置管理类
class HBaseConf {
private static HBaseConfiguration HBaseConfig=null;
static Configuration conf = null;
```

```
static {
conf = HBaseConfiguration.create();
conf.set("HBase.master", "192.168.1.2");
conf.set("HBase.zookeeper.property.clientPort","2181");
conf.set("HBase.zookeeper.quorum","192.168.1.2,192.168.1.21, …");
}
}
//HBase 数据库表创建管理类
class HBaseCreate {
public static void createTable(String tableName,String familys) throws Exception{
try {
HBaseAdmin HBaseAdmin = new HBaseAdmin(HBaseConf.conf);
if (HBaseAdmin.tableExists(tableName)) {
    System.out.println("table"+ tableName + "already exists");
}
else{
    HTableDescriptor tableDescriptor = new HTableDescriptor(tableName);
    tableDescriptor.addFamily(new HColumnDescriptor(familys));
    HBaseAdmin.createTable(tableDescriptor);
}
}
catch(MasterNotRunningException e){
    e.printStackTrace();
}
catch(ZooKeeperConnectionException e){
    e.printStackTrace();
}
catch(IOException e){
    e.printStackTrace();
}
}
}
```

HBase 数据查询管理的实现如下。

```
//查询 HBase 数据库表中某行某列的值
public static void getoneQualifier(String tableName,String rowKey,String
column,String qualifier) throws IOException
{
if (!tableName.equals("")) {
```

```
    if (!rowKey.equals("")) {
        if (!column.equals("")) {
            if(!qualifier.equals("")) {
                try{
                HTable table=new HTable(HBaseConf.conf,tableName);
                Get get = new Get(Bytes.toBytes(rowKey));
                get.addColumn(Bytes.toBytes(column), Bytes.toBytes(qualifier));
                Result r=table.get(get);
                byte[] val=r.getValue(Bytes.toBytes(column), Bytes.toBytes(qualifier));
                    System.out.println("rowKey: "+new String(r.getRow())+"
                 qualifier"+qualifier+" value: "+Bytes.toString(val));
                }//try
                catch (IOException e) {
                    // TODO Auto-generated catch block
                    System.out.println(e.getMessage() + "scan failed");
                }
            }//if
            else{
                System.out.println("the qualifier is null,please enter the ensure
qualifier");
            }
        }//if
        else{
            System.out.println("the family is null,please enter the ensure family");
        }
    }//if
    else{
        System.out.println("the rowkey is null,please enter the ensure rowkey");
    }
   }//if
   else{
    System.out.println("the table is null,please enter the ensure table");
   }
   }
```

9.4.4 基于 MapReduce 的数据转换管理模块

基于 MapReduce 的数据转换管理结构设计如图 9.28 所示。

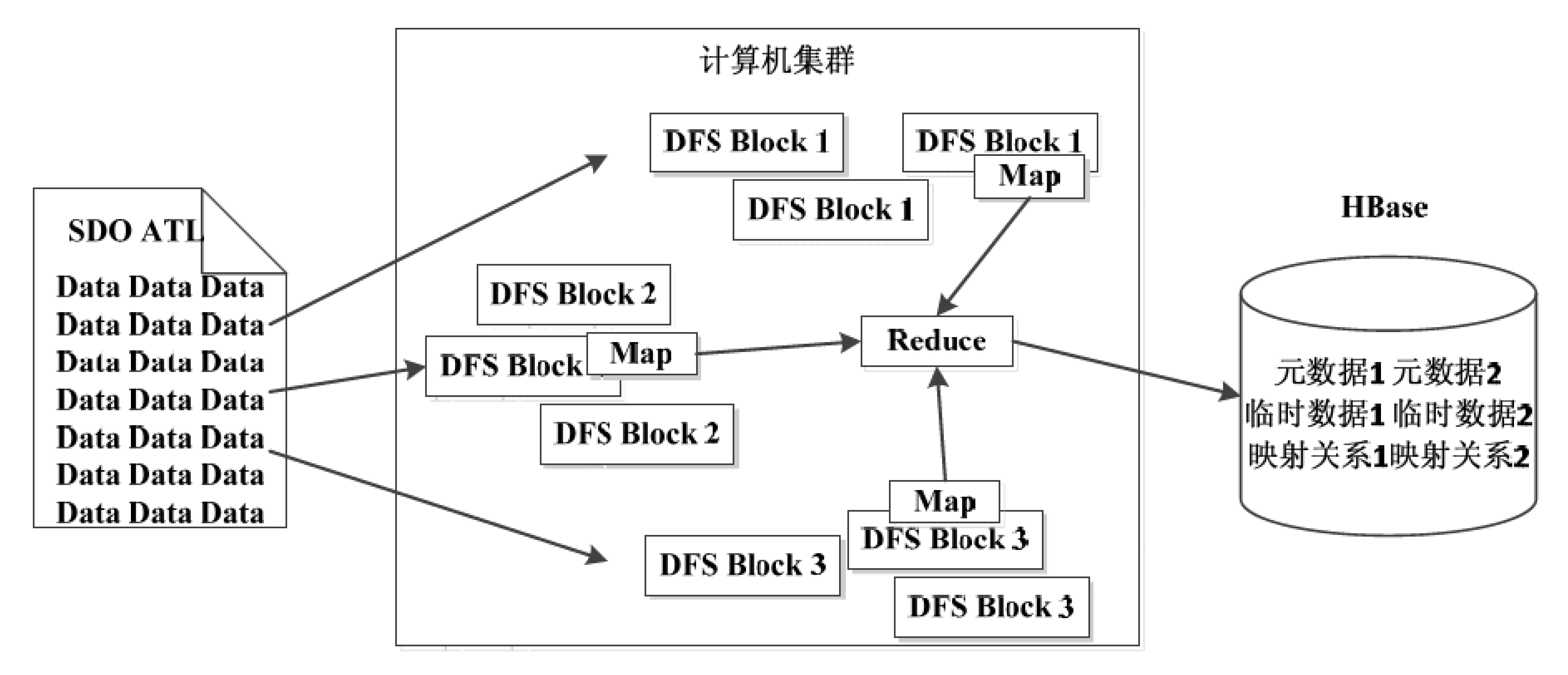


图 9.28　MapReduce 数据转换管理结构

首先介绍基于模型转换语言 ATL 的数据转换过程。在 ATL 模型转换的层次结构中需要 4 部分：元源模型（MMa）、元目标模型（MMb）、源模型（Ma）以及模型转换模型（Mt，也是模型转换的核心规则定义）。Ma 符合其元模型 MMa，而 Mb 符合其元模型 MMb。同时 MMa 和 MMb 都符合唯一的元元模型 MMM。Mt 是一个模型转换实例，符合模型转换的元模型 MMt。MMt 符合唯一的元元模型 MMM。MMt 通过 ATL 定义，Mt 需要用户依据具体需求定义实现转换规则。

基于 MapReduce 的 ATL 数据转换内部结构如图 9.29 所示。

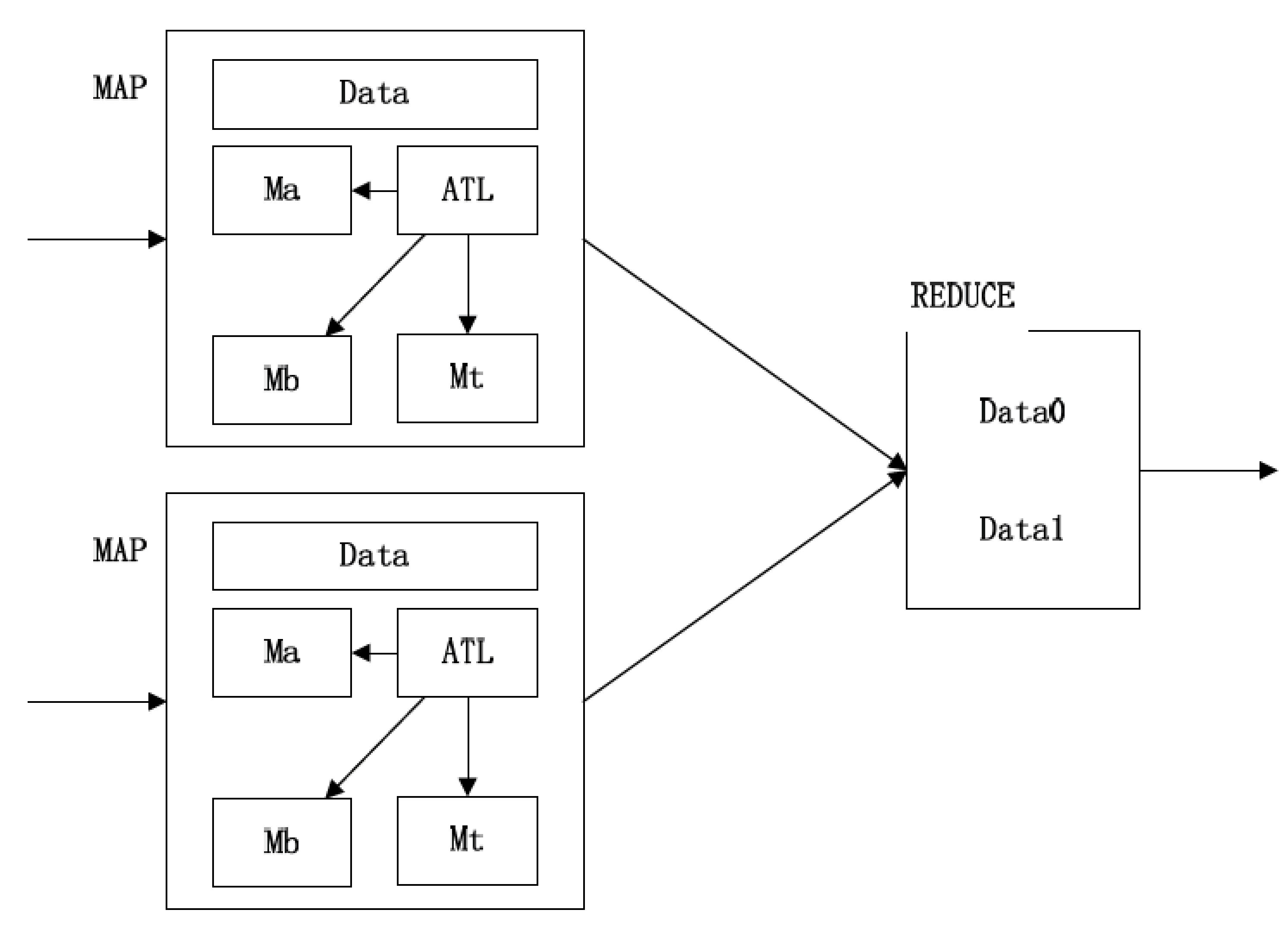


图 9.29　数据转换内部结构

图 9.29 中 Map 部分实现海量数据的分布处理。数据转换过程中在 Map 阶段加载 Ma、Mb、Mt 和数据并实现数据的转换，转换结束后通过 Reduce 过程实现目标数据的合并。

下面以简单的代码来描述一个数据转换实例的部分实现。

Map 阶段数据加载与转换伪代码实现如下：

```
Map()
{
```

```
        LoadMa(Ma);
        LoadMb(Mb);
        Mt=LoadMt(Ma,Mb);
        LoadData(Data);
        Transform(Ma,Mb,Mt,Data);
    }
```

元源模型 Ma 以及元目标模型 Mb 的描述如下面的代码段所示：

```
package Ma
{
class Ma
     {
   attribute name : String;
   attribute surname: String;
    }
}
package PrimitiveTypes
{
  datatype String;
}
```

元目标模型 Mb 的简单描述如下面的代码所示：

```
package Mb
{
  class Mb
     {
   attribute name : String;
   attribute surname: String;
    }
}
 package PrimitiveTypes
{
  datatype String;
}
```

将上述两个元模型经过转换生成基于 XMI 格式 Ecore 模型的元源模型 Ma.ecore 和元目标模型 Mb.ecore。

待转换源模型数据的实现如下：

```
Ma.ecore:
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Ma">
  <Ma name="David" surname="Touzet"/>
  <Ma name="Freddy" surname="Allilaire"/>
</xmi:XMI>
```

数据转换规则 Mt 伪码描述如下：

```
create OUT : Mb from  IN : Ma;
uses strings;
rule Author{
 from
   a: Ma! Ma
 to
   p: Mb! Mb(
      name<-a.name,
      surname<-a.surname
   )
}
```

在上述基于规则的代码中简单的定义了两个模型之间的转换规则映射。接下来讲述在 MapReduce 计算模型中实现上述数据转换的设计思想。

通常情况下，MapReduce 程序的编写都可以简单地依赖于一个模板及其变种。使用这个模板编写 MapReduce 程序，主要需要实现两个函数：继承自 Mapper 的 map 函数以及继承自 Reducer 的 reduce 函数。一般遵循以下格式：

```
Map:  (k1,v1) -> list(k2,v2)
public static class Map extends Mapper<K1, V1, K2, V2>
{
 public void map(K1 key, V1 value, Context context)
      throws IOException { }
}
Reduce: (k2,list(v2)) ->list(v3)
public static class Reduce extends Reducer<K2, V2, K3, V3>
{
 public void reduce(K2 key, Iterator<V2> values, Context context)
      throws IOException { }
}
```

对于海量数据的转换来说，可以将待处理的数据按照数据源来源（IP 地址和端口号）划分，

并且都可以完全并行地进行处理。

基于 MapReduce 的数据转换类关系如图 9.30 所示.

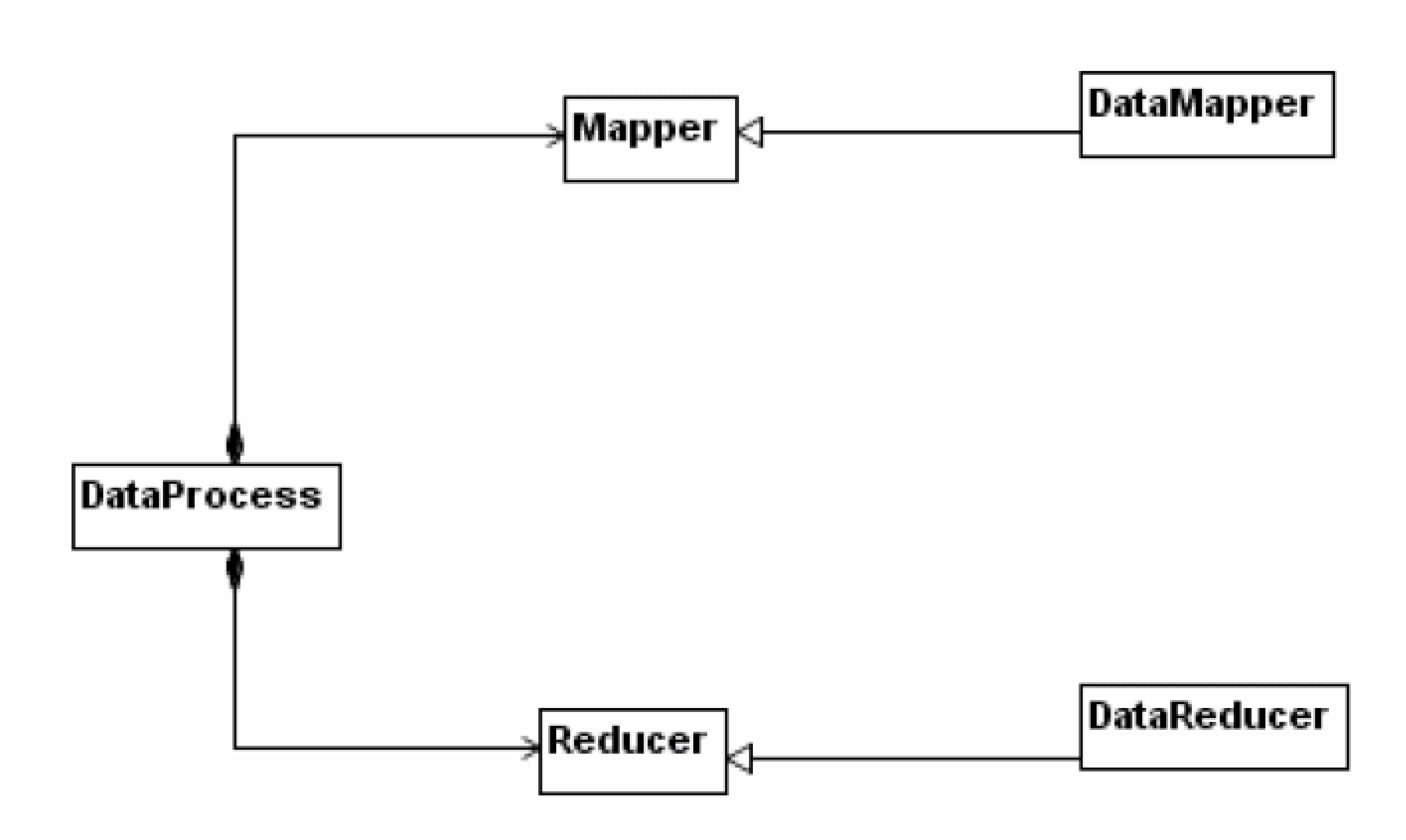


图 9.30 MapReduce 数据转换类关系

- DataMapper: 用户 Mapper 类。该类继承 Mapper 类，并重写 Mapper 类中的 map 函数，主要实现数据的 map 操作。Map 的输入主要包含元源模型、元目标模型、数据源数据以及数据转换规则，对输入的 value 值按数据源数据处理，并将数据按照转换规则转换等信息。
- DataReducer: 用户 Reducer 类。该类继承了 Reducer 类，并重写 Reducer 类中的 reduce 函数，实现对 Map 计算的数据结果按数据源来源合并进而提供给用户应用或存储在目标数据库中（如 HBase）。

DataMapper 和 DataReducer 是在 MapReduce 计算模型实现数据转换中最重要的两个类，分别实现了 MapReduce 编程模型中的 map 操作和 reduce 操作。用户可以通过 Hadoop 提供的接口来实现基于 MapReduce 的高效计算。

9.5 本章小结

本章主要讲述了基于服务数据对象的异构数据统一访问访问与灵活转换。在介绍了服务数据对象及数据统一访问转换的应用背景后，由浅入深地详细讲解了数据的统一访问与转换应用背景，接下来首先结合应用需求对数据统一访问与转换进行了总体的分析和设计，然后介绍了数据统一访问与数据转换过程中涉及的关键技术，最后结合数据访问服务、服务数据对象技术以及 MapReduce 计算模型和 HBase 存储技术对异构数据的统一访问访问和数据的转换各阶段过程做出详细的分析和设计。

第三篇

大数据应用篇



第 10 章

基于微博的股票市场预测系统

本章主要介绍了一种基于微博的股票市场预测系统，并对该系统的应用背景和相关技术做了相应的介绍，分析了系统的需求分析与总体设计思路，最后给出了详细的设计与实现方案。本系统主要是以 Twitter 微博为例，通过获取海量微博数据，构建股票兴趣用户网络，并分析用户的情感态度与用户之间的影响关系，预测股票看涨和看跌情感在整个社交网络中的传播趋势，进而预测股票价格在未来某时间段内的涨跌趋势。

10.1 应用背景介绍

社交网络即社交网络服务，它源自于英文 SNS（Social Network Service）的翻译，中文直译为社会性网络服务或者社会化网络服务，意译为社交网络服务。它源自网络社交，通过网络这个载体把人们连接起来，从而形成了具有某一特点的团体。Web 2.0 技术的兴起将人与人的交往全面带入了一个在线社交的时代。微博（Micro-Blog）、社会书签（Socialbookmark）、博客（Blog）、论坛（Forum）、维基（Wiki）、播客（Podcast）、协同内容标注（Collaborative tagging）等各类在线社交类应用迅速增长，用户利用这些社会网络应用完成比传统方式更直接的交互活动：社会书签中的标注、群体和推荐为不同用户间的知识共享提供更直接的支持；论坛站点为话题讨论提供了更有效的信息发布、浏览、分类；博客和微博为用户提供了简便的信息发布平台；维基网站将用户协同编辑和信息发布融为一体；博客和协同内容标注网站则将音频、图像、视频等多媒体内容进行发布并提供知识共享和信息传播功能。近年来，以 Twitter、Facebook、Linkedin、开心网、人人网、新浪微博等网站为代表的新一代互联网迅速崛起。这些社交应用都有一些共同的特点，例如用户贡献数据、利用群体智慧、充分的在线交互以及丰富的用户体验等。

Twitter 作为一种特殊的在线社交网络，引领了微博潮流，它凭借平台的开放性、终端扩展性、内容简洁性和低门槛等特性，成为网民获取新闻时事、人际交往、自我表达、社会分享以及社会参与的重要媒介。微博正在发展成为重要的新闻源，使新闻媒体的传播形态发生变化。Twitter 等在线社交网络与传统的 Web 网络在信息传播方式上存在着根本的不同：传统的 Web 网络是以信息内容为主体进行传播；在线社交网络是以人为中心，依靠人与人之间的好友关系进行信息的传播。微博打破了传统的传播方法，它利用人与人的关系改变人与信息的关系，反过来又用人与信息的关系影响人与人的关系。如何去描述在线社交网络中的信息传播行为，揭示它的特性，建立信息传播模型，具有重要的理论和应用价值。

这种对于自己想法的主动发布与分享，使得我们又有了一个平台，探究用户的想法，并根据其想法预测其进一步的行动。已经有学者开始围绕 Twitter 微博平台进行一些预测方面的研究。他们认为，通过对 Twitter 上谈及某特定事件的微博的内容、发送数目、发送时间和频率等信息的研究，能够预测一些特定事件宏观上的走向，目前这类研究涵盖突发事件处理、电影票房统计等多方面的信息。近年来我国网络大事层出不穷，网络舆论一直处于井喷状态，在这一背景下，如何尽早地预测社会网络上话题的走向和趋势对于国家政治稳定具有重要的现实意义。在商业推广领域，充分利用社会网络的小世界特性，在有限节点上进行合理的广告投放，推动内容演化，往往可以在满足商业需求的前提下，最大限度地减少商业成本，从而达到良好的产品推广效果，因此，近年来社会网络上的广告推广也受到了广泛的关注。研究信息在社交网络中的传播，并对趋势分析做出合理预测，不但对国家政治稳定具有重要的现实意义，而且在商业应用上也有良好的前景。

例如，2011 年轰动一时的“郭美美事件”，起因是一位微博昵称“郭美美 baby”的人在网上公然炫耀其奢华的生活，并称自己是中国红十字会商业总经理。她的炫富微博起初只被一小部分网友看到并转发，而后便大规模地被转发并且评论，相关发言迅速增长到了 64 余万条，一时之间引发了部分网友对中国红十字会的非议，成为当时的社会热点话题。这一事件让我们看到了微博的巨大力量。另外，微博信息趋势预测的研究前景方向极广，越来越多的科研项目都开始运用微博上的数据，进行对包括金融、医学、地震以及人的情绪的研究，例如在金融界的病毒式营销（Viral Marketing）模式。当我们想为一个新产品做推广的时候，倘若已经知道一个社交网络中个体间的影响能力且已知一些最有影响力的用户，那么可以通过用户和用户之间的“口口相传”得到一连串的连锁反应，从而达到网络中比例相当大的一部分用户并且让他们了解到这个新产品。一些侧重于“影响最大化”的数据挖掘问题就是在研究这类问题并将它转化为“如何从 k 个节点中选出一个初始集从而使得网络最后有最多的节点能受到影响”的问题。又例如在医学界，一些科学家追踪了 2009 年和 2010 年期间关于猪流感的微博，并研究了这些相关的微博与疫苗接种率的关系。通过比较 Twitter 数据和疫苗接种率，美国疾病预防与控制中心发现了人们对流感疫苗的看法与他们是否得病之间的关联模式。举个例子，新英格兰是美国疫苗接种率最高的地区，也是最多 Twitter 用户发布关于疫苗接种的积极信息的地区。这项科研结果发表在《PLoS 生物学》期刊上。医护人员可以利用该项数据来预测疾病的爆发，且能更迅速地对流行病做出反应。还有很多类似的研究也已经开展。

针对以上背景，作者致力于研究一种基于微博的信息传播预测模型，对网络中的多种话题进行传播预测，并将基于微博的信息传播预测模型创新地应用到股票价格涨跌预测中。研究表明，微博作为社交网络的一个产物，已成为一个充满活力的在线交流平台，用户每天在微博上产生海量的数据并在用户之间产生大量的信息流传播。据统计，每天大约有 25 万与股票相关的微博消息，通过计算语言学的方法对这些消息进行分析，研究发现微博的情绪（比如看涨）能够在一定程度上影响股票日交易量和未来股票的涨跌。从而达到股票预测和分析的目的。一些调查也揭示了微博情绪与股市关联的冰山一角，如当在世界杯、欧洲杯的国际球赛中输球时，微博平台上很快出现很多负面情绪，当地的股市指数也会受到显著的负面影响。

然而，目前研究人员对于社交网络的研究面临着相当多的挑战。由于微博中的信息量巨大，信息的种类和话题繁多，并且消息的传播速度也很快，这给相关的研究带来了很大的困难和巨大的工作量。因此，需要一个可靠的算法来帮我们从海量的数据中提取信息并且能够了解信息的传

播方式，进而预测未来一段时间内股票市场的趋势。本系统通过对 Twitter 上捕获的大量信息进行过滤、分类和分析，应用构建好的社交网络结构，分析用户之间的相互影响和信息传播路径，从而预测某支股票在未来一段时间内的涨跌趋势。这对于帮助投资者预测和分析股市，选择股票进行投资，优化组合投资，降低投资风险，获得最大收益具有重大意义。

10.2 需求分析与总体设计

10.2.1 需求分析

本系统是基于微博的股票市场预测，作者以 Twitter 为平台来进行数据采集、分析和预测研究。在具体实现中需要根据 Twitter 社交网络的特点，提取股票相关的微博内容，建立股票兴趣用户网络，分析网络中用户之间的相互影响情况，预测用户对股票的看涨情感以及看跌情感在整个社交网络中的传播趋势，进而可以预测该股票在未来某时间段内的价格涨跌趋势。股票市场预测系统的框架图如图 10.1 所示。

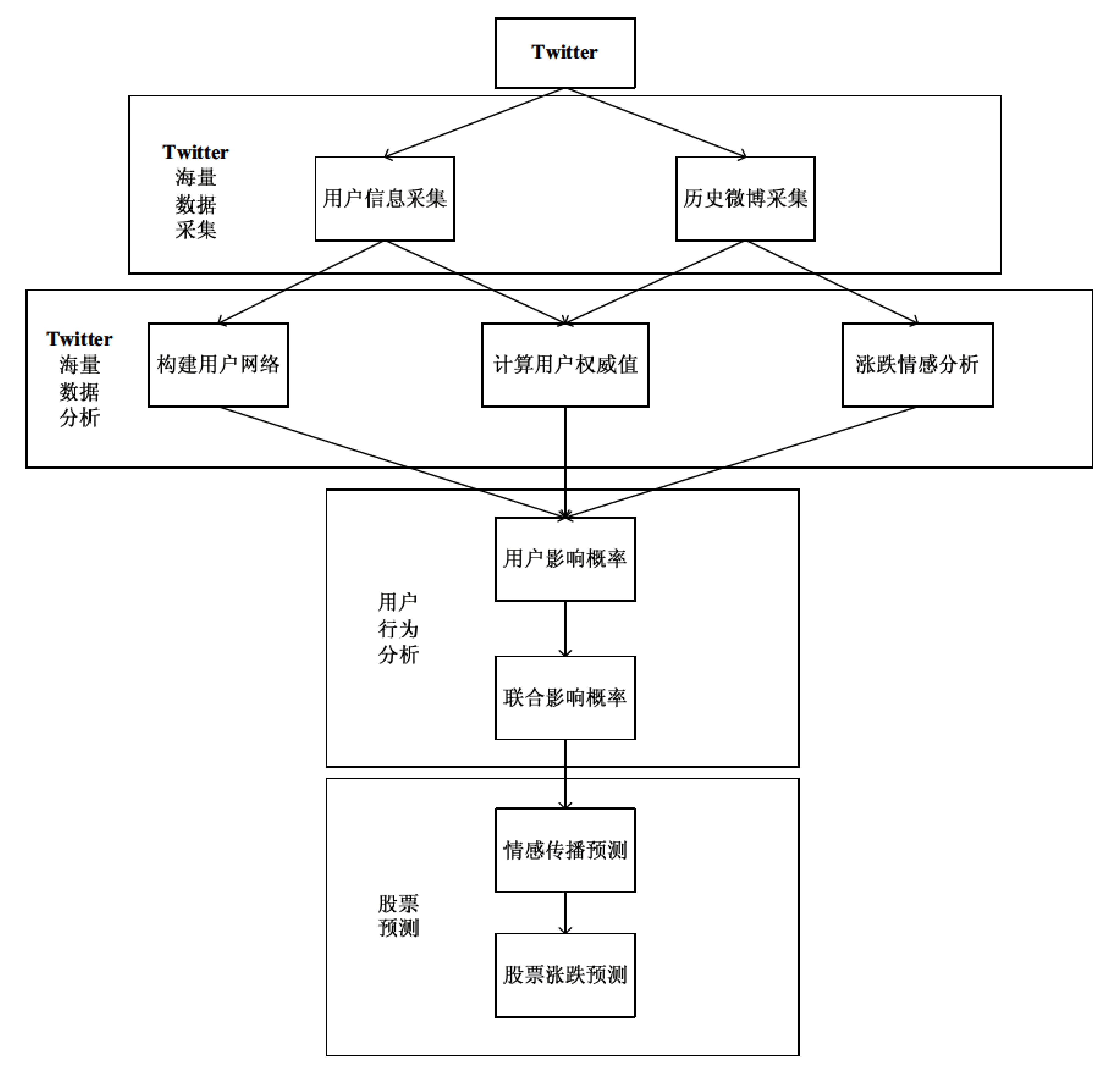


图 10.1　股票市场预测系统框架图

从系统框架图可以看出，股票市场预测系统主要由 Twitter 海量数据采集、Twitter 海量数据分析、用户行为分析、股票预测 4 部分构成。其用例图如图 10.2 所示。

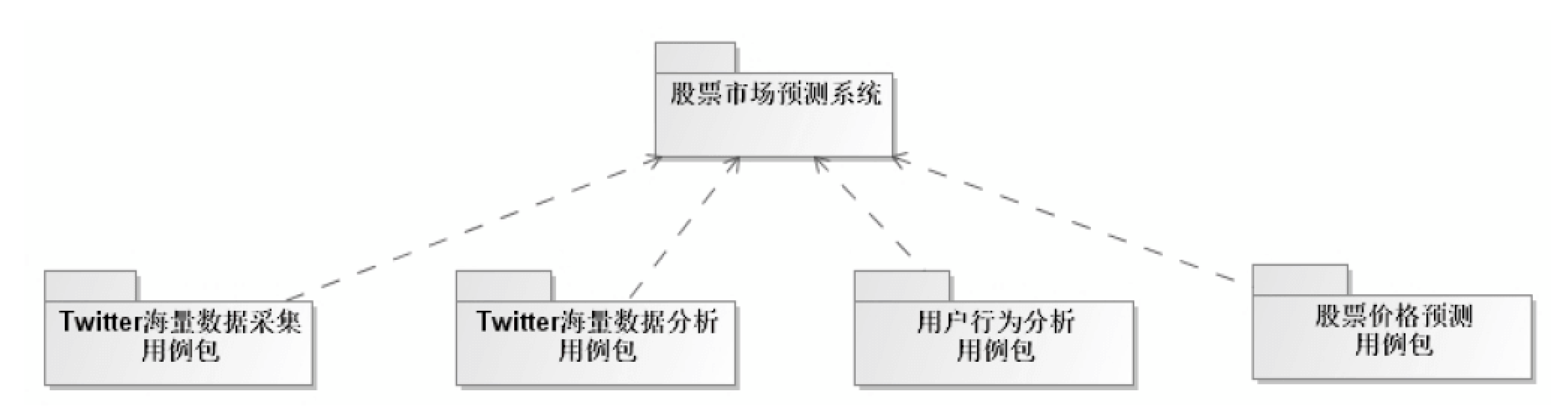


图 10.2　股票市场预测系统用例包

（1）　Twitter 海量数据采集

数据采集是该系统的基础，用户需要通过该系统能够高效率地对海量文本进行抽取，并且能够及时存入数据库中。高效及时的采集数据是实现该系统的前提。数据采集包括两个基本用例：历史微博采集和用户信息采集。图 10.3 为 Twitter 海量数据采集用例图。

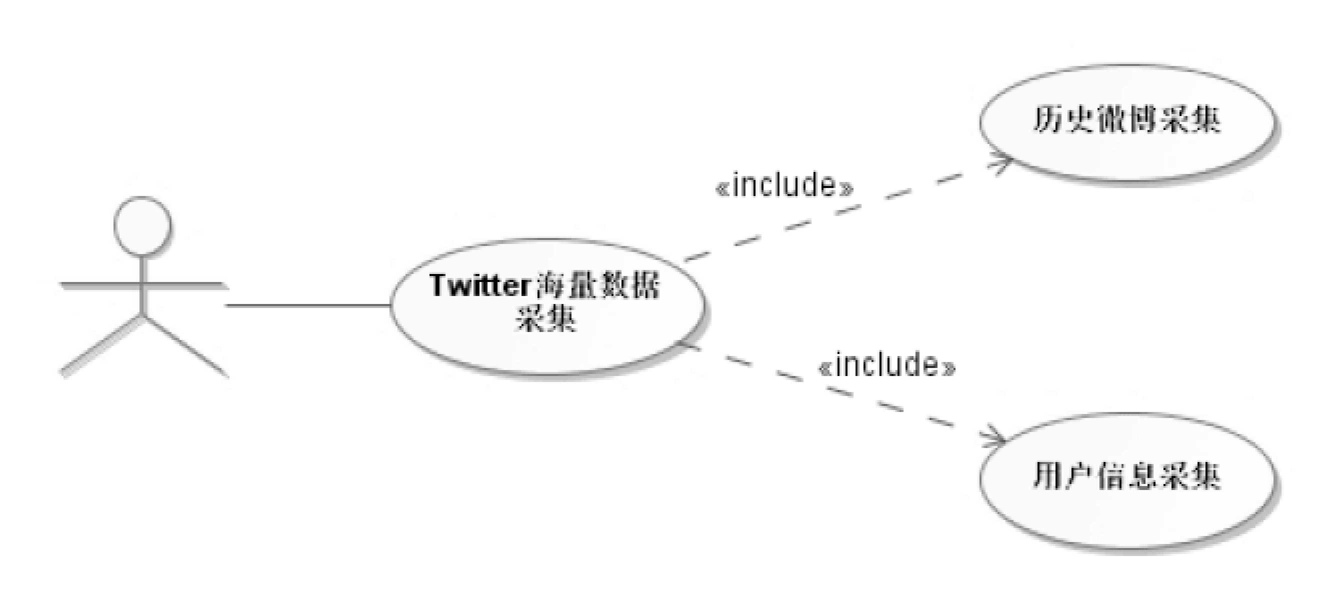


图 10.3　Twitter 数据采集用例图

（1）　历史微博采集

在该用例中，系统能够根据特定的关键字，在 Twitter 中进行搜索，对搜索的结果进行处理和解析，提取系统需要的信息，包括微博的 id、内容、发布时间、发布作者 id 等，然后将这些信息存入数据库中。本系统是对股票市场的预测，所以需要采集大量有关股票的微博，在 Twitter 中人们谈论某支股票时往往以“$”+股票缩写为关键字，这为采集提供了很大的便利。表 10.1 为历史微博采集详细用例表格。

表 10.1　历史微博采集详细用例表

用例编号	UC-1-1
用例名称	历史微博采集
参与者	用户、预测系统
描述	该用例描述了系统采集历史微博的过程。根据用户需要采集某支股票的关键字，系统通过 Twitter 提供的搜索功能，获取有关该关键字的所有微博消息，对海量的微博数据进行解析，获取包括微博的 id、内容、发布时间、发布作者 id 等数据，然后将这些信息存入数据库中

（续表）

用例典型事件流	参与者动作	系统响应
	Step1：输入搜索的股票关键字	
		Step2：判断该关键字是否输入正确
		Step3：系统向 Twitter 发出请求，搜索并获取该关键字的所有微博消息
		Step4：系统对获取的微博进行处理，提取微博的 id、内容、发布时间、发布作者 id 等内容
		Step5：系统将提取的海量微博数据存入数据库中
	Step6：返回成功获取的历史微博信息	
可选事件流	Step2，如果搜索的关键字不是有关股票内容的，系统将返回“获取信息失败”提示信息	
前置条件	用户已获取需要搜索和采集的股票代码	
后置条件	用户可以通过数据库查询采集到的历史微博信息	
假设	无	

（2）用户信息采集

在该用例中，系统能够根据用户的 id 或 name 抽取到特定用户在 Twitter 上的基本注册信息，包括 id、name、language、description、following、follower 等。following 是指该用户在 Twitter 上所关注的那些用户 id 集合，follower 是指关注该用户的那些用户 id 集合。这里系统需要采集的用户是指发布了有关谈论股票微博的那些用户。表 10.2 为用户信息采集详细用例表。

表 10.2　用户信息采集详细用例表

用例编号	UC-1-2	
用例名称	用户信息采集	
参与者	预测系统	
描述	该用例描述了系统采集用户信息的过程。根据某个用户的 id 或 name，系统通过 Twitter 获取该用户的 id、name、language、description、following、follower 等信息，并将返回结果存入数据库中	
用例典型事件流	参与者动作	系统响应
	Step1：输入用户 id 或 name	
		Step2：确认该用户已在 Twitter 注册
		Step3：判断该用户信息是否已存入系统数据库中
		Step4：系统向 Twitter 发出请求，获取该用户基本信息，包括 id、name、language、description、following、follower 等
		Step5：系统获取采集到的用户信息，并添加到数据库中
	Step6：返回成功获取的用户信息	

（续表）

可选事件流	Step2，如果需要采集的用户在 Twitter 中不存在或已过期，系统将返回“获取用户信息失败”提示信息。 Step3，将用户 id 在数据库中查询，如果发现该用户已存入数据库，则系统不再向 Twitter 发出请求，直接返回“该用户已存在”提示信息
前置条件	系统已通过采集到的历史微博，提取出该微博的用户 id，并将该 id 作为需要采集用户的 id 输入
后置条件	用户可以通过数据库查询采集用户的基本信息
假设	无

2. Twitter 海量数据分析

数据分析是该系统的关键，系统能够对采集到数据库中的海量大数据进行处理和分析，提取出为下一步用户行为分析和股票涨跌预测有用的关键数据。由于存储的是海量大数据，这就需要系统能够高效率地对海量数据进行数据库的读写和分析处理。数据分析包括三个基本用例：构建用户网络、计算用户权威值、涨跌情感分析。图 10.4 为 Twitter 海量数据分析用例图。

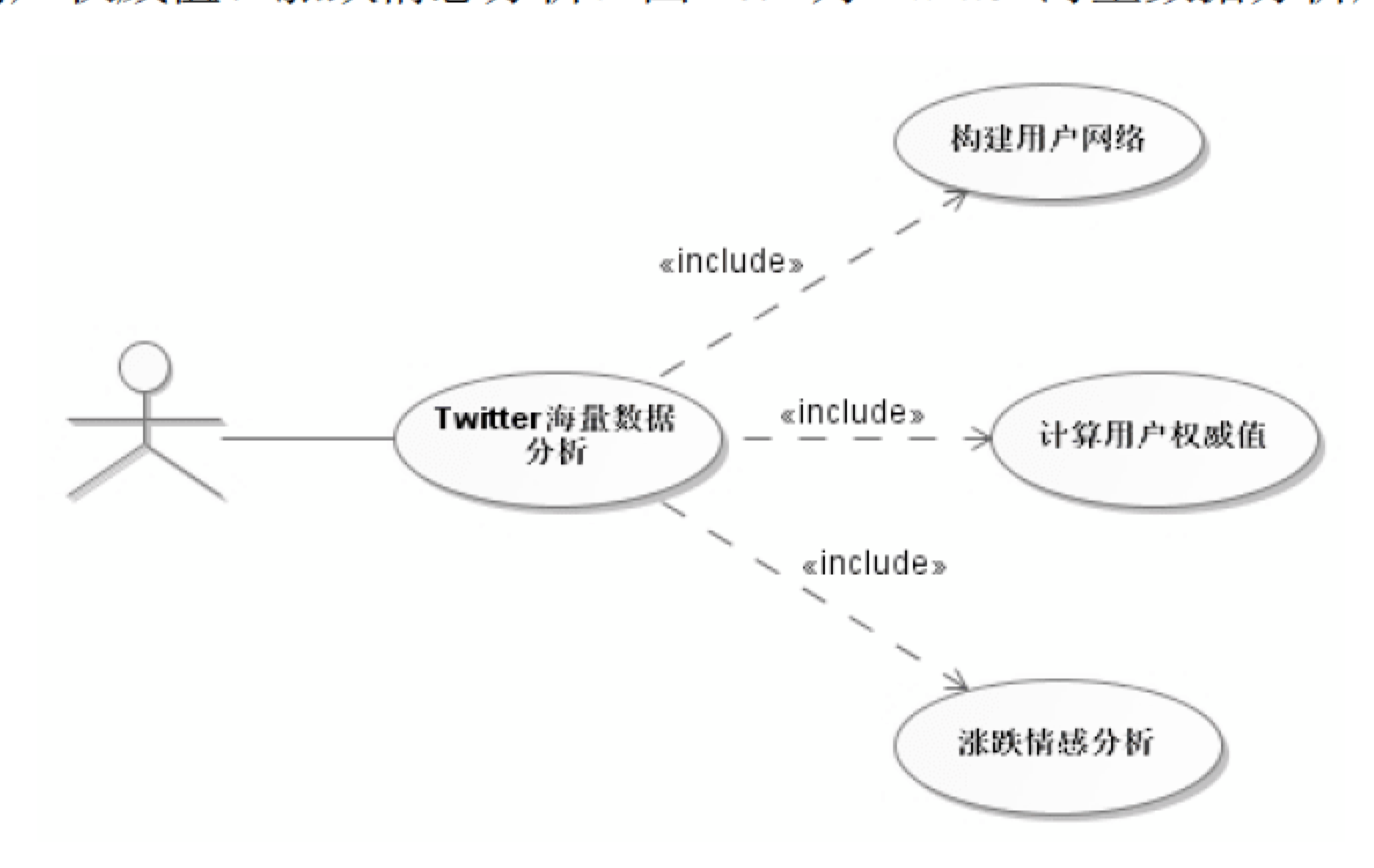


图 10.4 Twitter 数据分析用例图

（1）构建用户网络

在该用例中，系统根据采集到的用户信息，为每一支股票构建一个社交网络结构图，在该结构图中的所有用户都会经常谈论这支股票，并且发表有关该股票的微博信息。由于用户网络图中每个用户都不是孤立存在的，他们需要 following 其他用户，或被其他用户 following，这样就构建成了一个有向图。如图 10.5 所示，图中的箭头表示用户之间的 following 关系。系统通过读取数据库中采集到的用户基本信息，判断一个用户的 followers 和 followings，构建基本的用户网络图。表 10.3 为构建用户网络详细用例表。

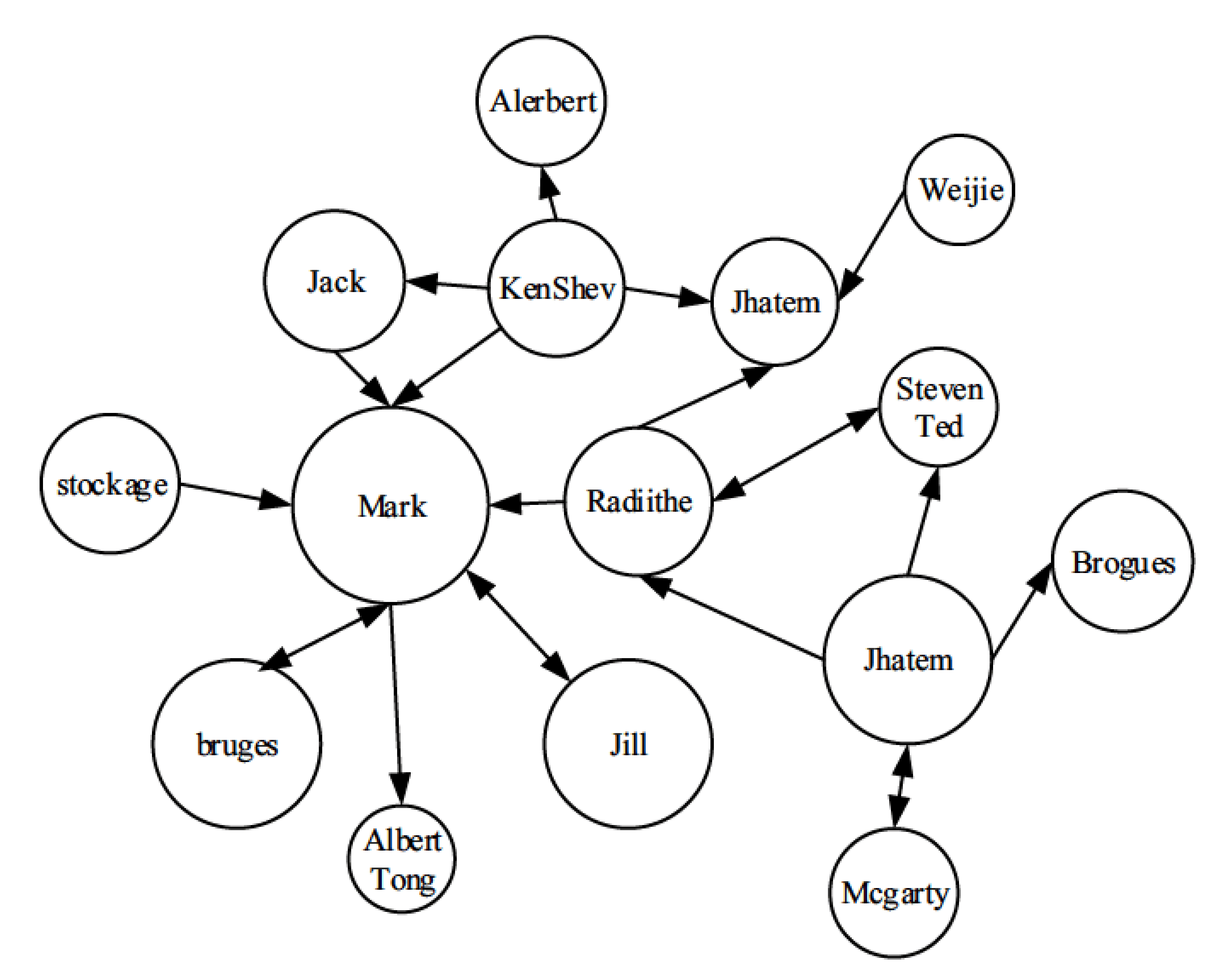


图 10.5　用户网络结构图

表 10.3　构建用户网络详细用例表

用例编号	UC-2-1	
用例名称	构建用户网络	
参与者	预测系统	
描述	该用例描述了系统构建用户网络的过程。根据数据库中采集的用户信息，每一个用户都有自己的 following 集合和 follower 集合，系统读取每个用户相应的集合，判断用户之间是否有 following 关系，构建用户网络结构图，following 关系用图中的有向边表示。最后，将构建好的图结构存入数据库中	
用例典型事件流	参与者动作	系统响应
	Step1：读取数据库用户信息	
		Step2：查询该用户的 following 和 follower 集合
		Step3：根据用户 following 关系构建有向图中的边
		Step4：系统将构建的有向图存入数据库中
	Step5：返回网络图结构	
可选事件流	Step2、Step3，如果该用户不存在 following 或 follower 集合，系统将不把该用户作为图中节点存在。相应地也不再构建有关该用户节点的边信息	
前置条件	系统将用户基本信息存入数据库中，并且能够查询每个用户的 follower 和 following 集合	
后置条件	用户可以查询构建的网络结构图，并判断任意用户之间的 following 关系	
假设	无	

（2）计算用户权威值

该用例是在系统内部实现的，在该用例中，系统对采集到的用户历史微博进行处理和分析，并结合用户自身的一些基本信息，计算出该用户在社交网络结构中的权威值。所谓权威值是指该用户在社交网络中的影响力和言论的分量，权威值高的用户，对社交网络中子节点的影响就大，

体现在对子节点的影响概率数值上就会更大一些，举例来说，美国总统奥巴马（@Barack Obama）在 Twitter 平台上的影响力与普通 Twitter 用户的影响力自然有明显区别。在图 10.5 中用户节点的大小表示了该用户的权威值，权威值越高则节点越大，反之，则节点越小。表 10.4 为计算用户权威值详细用例表。

表 10.4　计算用户权威值详细用例表

用例编号	UC-2-2	
用例名称	计算用户权威值	
参与者	预测系统	
描述	该用例描述了系统计算用户权威值的过程。系统根据数据库中采集到的用户历史微博，对该用户所发布的微博进行分类与处理，并结合用户的基本信息，计算出该用户在社交网络结果中的权威值，并将该权威值作为用户属性的一部分存入数据库中	
用例典型事件流	参与者动作	系统响应
	Step1：读取数据库中用户历史微博和用户基本信息	
		Step2：对用户历史微博进行分类和处理
		Step3：提取用户基本信息
		Step4：计算该用户的权威值并存入数据库中
	Step5：获取并查看该用户在网络结构中的权威值	
可选事件流	Step4，如果数据库中用户历史微博进行了更新，则需要重新计算该用户的权威值，并及时更新该用户数据库中的权威值	
前置条件	该操作是系统内部发出的	
后置条件	可以通过数据库查询计算出的用户权威值	
假设	无	

（3）涨跌情感分析

在该用例中，系统对提取到的历史微博进行预处理，由于采集到的历史微博都是谈论股票的，那么每一条微博相对于某支股票都会有一定的情感态度，我们按看涨、看跌和持平（波动）三种情感进行分类。通过本规范化、分词、去停止词、绑定否定词、词典匹配，对微博的情感进程计算，并将计算结果存入数据库。这里需要注意的是，由于波动情感在判断上的复杂度，为了简单起见，系统只分析股票微博的看涨和看跌情感。表 10.5 为涨跌情感分析详细用例表。

表 10.5　涨跌情感分析详细用例表

用例编号	UC-2-3	
用例名称	涨跌情感分析	
参与者	预测系统	
描述	该用例描述了系统对微博内容进行情感分析的过程。系统根据数据库中采集到的历史微博，对微博内容进行规范化、分词、去停止词、绑定否定词、词典匹配，最终分析出该条微博的情感态度，并按看涨和看跌两种态度存入数据库中	
用例典型事件流	参与者动作	系统响应
	Step1：读取数据库中历史微博的内容	

（续表）

用例典型事件流		Step2：对微博内容进行规范化处理
		Step3：对微博内容进行分词、去停止词、绑定否定词
		Step4：对处理后的微博内容进行词典匹配
		Step5：按看涨和看跌两种态度计算出微博的情感，并存入数据库中
	Step6：获取并查看微博的情感	
可选事件流	Step2，如果微博内容已经达到规范化要求，则不进行规范化处理 Step3，如果微博内容中没有停止词、否定词，则不进行相应的处理	
前置条件	该操作是系统内部发出的	
后置条件	可以通过数据库查询计算出的微博情感	
假设	无	

3. 用户行为分析

用户行为分析是本系统的核心部分，通过对微博的预处理，系统根据得到的用户网络、用户权威值、微博情感分析等信息，对网络中用户的行为进行一定的分析，进而预测出未来一段时间内股票价格的涨跌情况。用户的行为分析主要包括两个用例：用户影响概率计算和联合影响概率计算。图 10.6 为用户行为分析用例图。

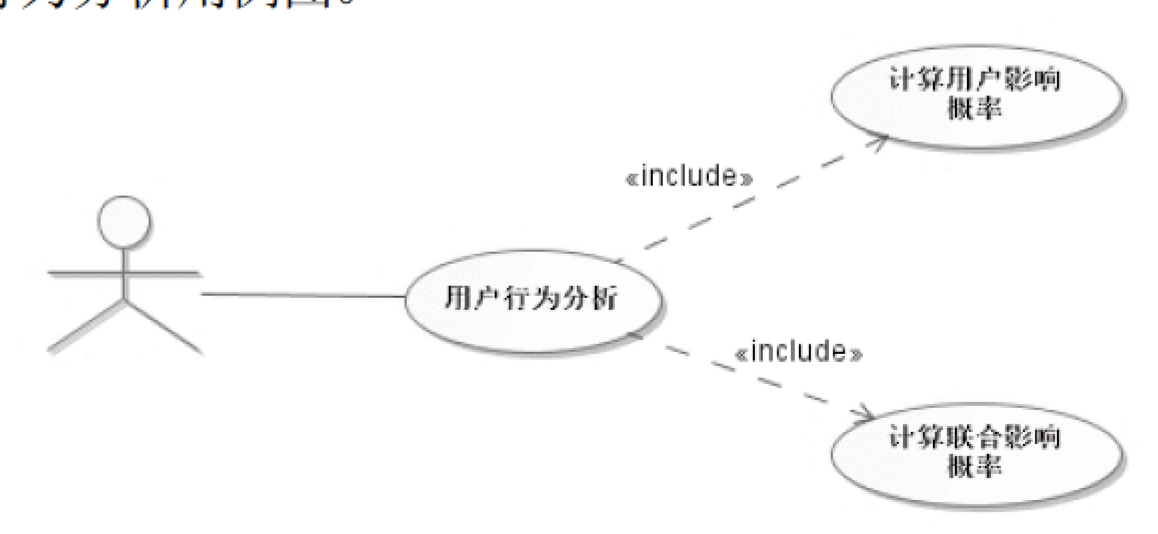


图 10.6　用户行为分析用例图

（1）　用户影响概率计算

在该用例中，系统根据用户发出的历史微博，结合用户之间的网络关系，可以计算出有相互联系的用户之间的依赖关系，这种依赖关系在本系统中称为用户影响概率，影响概率越大，表明该用户受到其他用户的影响越大，那么也会跟随这个用户发出相似的行为。反之，则表明该用户的行为不受其他用户的影响。在计算的时候，系统还会计算出一个平均影响时间，表明用户之间相互影响的持续时间为多长。表 10.6 为用户影响概率计算详细用例表。

表 10.6　用户影响概率计算详细用例表

用例编号	UC-3-1
用例名称	用户影响概率计算
参与者	预测系统
描述	该用例描述了系统对网络中用户之间影响概率的计算过程。系统根据采集到的用户历史微博，以及用户之间的网络关系，对用户受到其他用户的影响概率进行计算，并且计算出该用户之间的评价影响时间

（续表）

用例典型事件流	参与者动作	系统响应
	Step1：读取数据库中历史微博的内容	
	Step2：读取用户网络关系	
		Step3：计算并更新用户平均影响时间
		Step4：根据 Jaccard 系数连续时间模型计算用户之间的影响概率
		Step5：将用户之间的平均影响时间和影响概率存入数据库中
	Step6：获取并查看平均影响时间和影响概率	
可选事件流	Step3、Step4，如果用户的历史微博没有发出相似的行为，则不计算该用户的平均影响时间和影响概率	
前置条件	该操作是系统内部发出的	
后置条件	可以通过用户影响概率进而计算用户所受到的联合影响概率	
假设	无	

（2）联合影响概率计算

在该用例中，系统根据计算出的用户影响概率，进而可以计算出该用户受到周围所有父节点的联合影响概率。如果该用户所受到的联合影响大于一个阈值，则可以预测用户将会被父节点影响，从而成为传播节点，获得传染性，可以将行为继续传播到子节点中未被影响的用户节点。表 10.7 为联合影响概率详细用例表。

表 10.7　联合影响概率计算详细用例表

用例编号	UC-3-2	
用例名称	联合影响概率计算	
参与者	预测系统	
描述	该用例描述了系统对网络中用户所受到所有父节点的联合影响概率的计算过程. 系统根据计算出的单个用户之间的影响概率，通过自定义的联合影响概率公式，计算出该用户在网络中所受到的联合影响概率	
用例典型事件流	参与者动作	系统响应
	Step1：读取用户网络关系	
	Step2：读取数据库中该用户与周围父节点的用户影响概率	
		Step3：计算该用户所受到的联合影响概率
		Step4：将得到的联合影响概率存入数据库中
	Step5：获取并查看联合影响概率	
可选事件流	无	
前置条件	该系统已经计算出用户之间的影响概率	
后置条件	无	
假设	无	

4. 股票预测

股票预测是本系统的最后一部分，主要包括情感传播预测和股票涨跌预测两个用例。本系统主要根据计算出的用户行为分析和股票情感分析，预测出涨跌两种情感在用户网络中的传播趋势，判断出看涨和看跌的用户人数。进而根据“个人行为受情感支配”理论，可以预测这支股票在股市收市前是涨还是跌。图 10.7 为股票预测用例图。

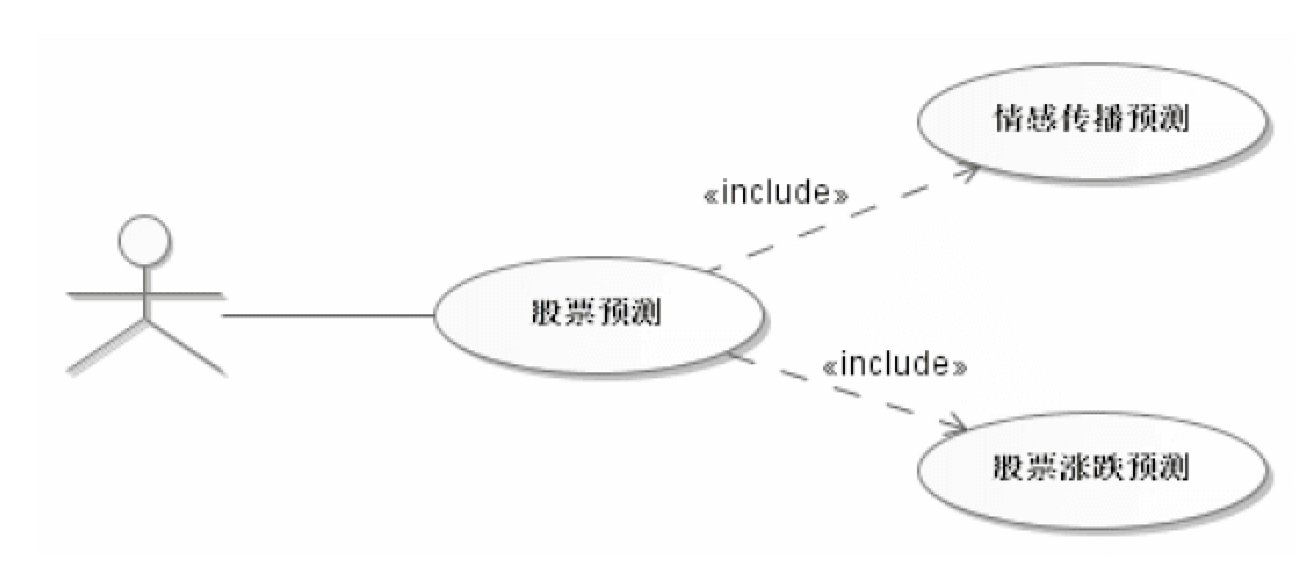


图 10.7　股票预测用例图

（1）情感传播预测

在该用例中，系统根据数据库中存储的微博情感和用户所受到的联合影响概率，可以预测出看涨和看跌这两种情感在用户网络中的传播趋势。在预测趋势的时候，系统会设置一个阈值，如果用户所受到的联合影响概率大于该阈值，则证明该用户会受到父节点情感的影响，成为新的传播节点。从而可以预测出整个网络中情感的传播趋势和感染人数。表 10.8 为情感传播预测详细用例表。

表 10.8　情感传播预测详细用例表

用例编号	UC-4-1	
用例名称	情感传播预测	
参与者	预测系统	
描述	该用例描述了系统对网络中情感传播预测过程的描述。系统根据阈值模型，设置一个传播阈值，以计算出的联合影响概率和微博情感为输入，分析出未来一段时间内看涨和看跌两种情感在网络中的传播趋势	
用例典型事件流	参与者动作	系统响应
	Step1：读取数据库中微博情感	
		Step2：按看涨和看跌两类情感对微博进行分类，并记录用户信息
	Step3：读取数据库中用户的联合影响概率	
		Step4：设定阈值，并根据阈值模型判断出情感的传播趋势，按看涨和看跌两类趋势进行预测
可选事件流	无	
前置条件	微博情感分析和联合影响概率已经计算完成	
后置条件	无	
假设	无	

（2）股票涨跌预测

在该用例中，用户需要选择需要预测的股票代码和时间，系统根据上一步分析的情感传播趋势，可以统计出预测的情感趋势，即预测的看涨人数和看跌人数，再加上之前情感分析的用户人数，系统将两部分人数进行统计，再根据“个人行为受情感支配”理论，可以预测这支股票在股市收市前是涨还是跌。表 10.9 为股票涨跌预测详细用例表。

表 10.9　股票涨跌预测详细用例表

用例编号	UC-4-2	
用例名称	股票涨跌预测	
参与者	用户、预测系统	
描述	该用例描述了系统预测股票价格涨跌的过程。用户输入需要预测的股票代码和时间，系统根据情感传播趋势来预测股票涨跌，并返回给用户最终预测结果	
用例典型事件流	参与者动作	系统响应
	Step1：输入需要预测的股票代码和时间	
		Step2：统计情感传播趋势中看涨和看跌的人数
		Step3：对未来股票价格的涨跌做出预测
		Step4：系统将预测结果返回给用户，并预测出看涨和看跌的比例
	Step5：用户查看预测结果	
可选事件流	Step2、Step3、Step4，如果用户输入的是无效的股票代码，则系统不能做出正常的响应	
前置条件	用户输入的股票代码必须是有效的，并且情感传播趋势已经分析完成	
后置条件	无	
假设	无	

10.2.2　总体设计

本系统通过分析 Twitter 这一社交网络平台，从海量的 Tweets 文本中提取出与股票相关的微博信息，同时构建社交网络，计算出用户在社交网络中的权威值，并对微博文本进行情感分析。根据社交网络关系和用户发出的历史微博，分析出用户之间的行为影响关系，从而可以预测出未来一段时间内，带有情感的微博在社交网络中的传播趋势。最后，根据用户情感的这种传播趋势，便可以预测出在未来一段时间内股票价格的涨跌趋势。由于在社交网络中用户之间是有影响和联系的，某个用户对某支股票的关注和态度可能会以一种“病毒”的形式在整个网路中传播和扩散，并且感染与他有关联的其他用户，从而影响其他用户对这支股票的情感态度，进而影响该用户的行为。另外，根据行为经济学研究表明，个人的情绪能够显著地影响个人行为和决策，社会网络中普遍传播的情感态度就可以影响整个社会的群体决策。针对股票市场，除了媒体新闻可以影响股票市场外，公共情感态度也起到了同等重要的作用。心理学研究就已经表明，除了信息可以影响人的决策之外，情感也是极其重要的。而行为经济学研究更是进一步认为，金融决策其实是受情感和情绪驱使的。本章介绍的系统就是根据这一原理设计的。

整个系统可以分成 4 个部分：海量微博数据采集、海量微博数据分析、用户行为分析和股票预测。

其中，海量微博数据采集的对象包括两部分内容，Twitter 上的用户信息和与股票相关的历史微博，然后，根据用户信息和股票历史微博，对海量数据进行分析，计算出用户权威值和微博涨跌情感。另一方面，用抓取的用户信息和用户发布的 tweets 构建社交网络，分析计算出用户之间的影响概率。进而通过用户之间的影响关系，预测出网络中的情感传播趋势和股票在未来一段时间内的涨跌趋势。

该系统可以分成以下几个部分：

- 通过 Twitter 的 API 获得用户数据以及有关股票的微博相关信息。
- Tweets 文本分析，根据用户发布的历史微博，分析用户在社交网络中的权威值；再对 Tweets 文本进行去噪声、分词、去停止词、匹配等处理，得到该条微博的情感态度。
- 分析用户之间的影响关系，计算用户之间的影响概率和用户所受到的联合影响概率。
- 根据用户受到的联合影响概率和某一时刻所有用户微博的情感，设计阈值模型，分析出未来一段时间内情感在网络中的传播趋势，进而预测出未来股票价格的涨跌趋势。

系统的流程图和数据流图如图 10.8 和图 10.9 所示。

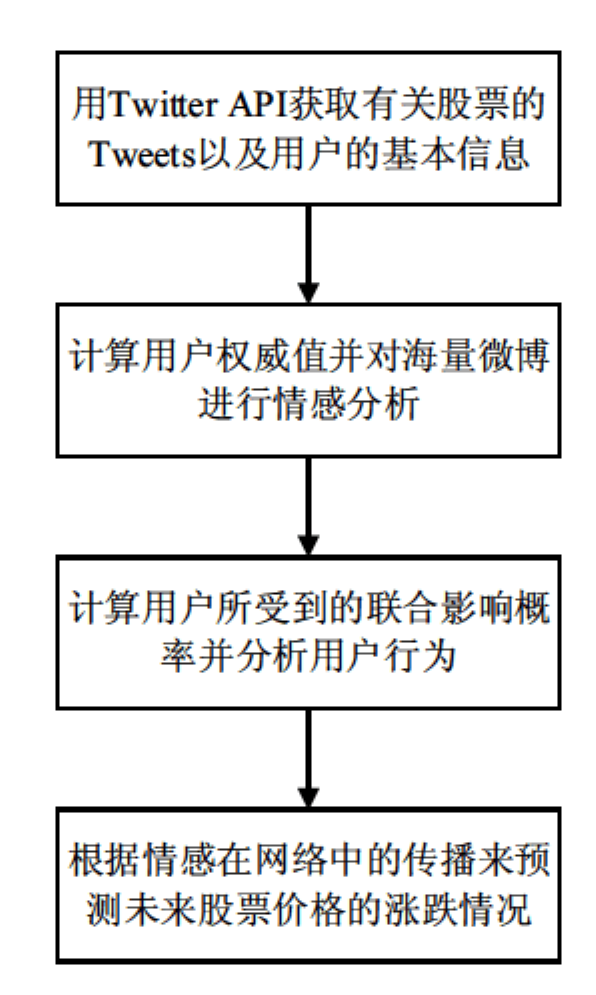


图 10.8　基于微博的股票市场预测系统流程图

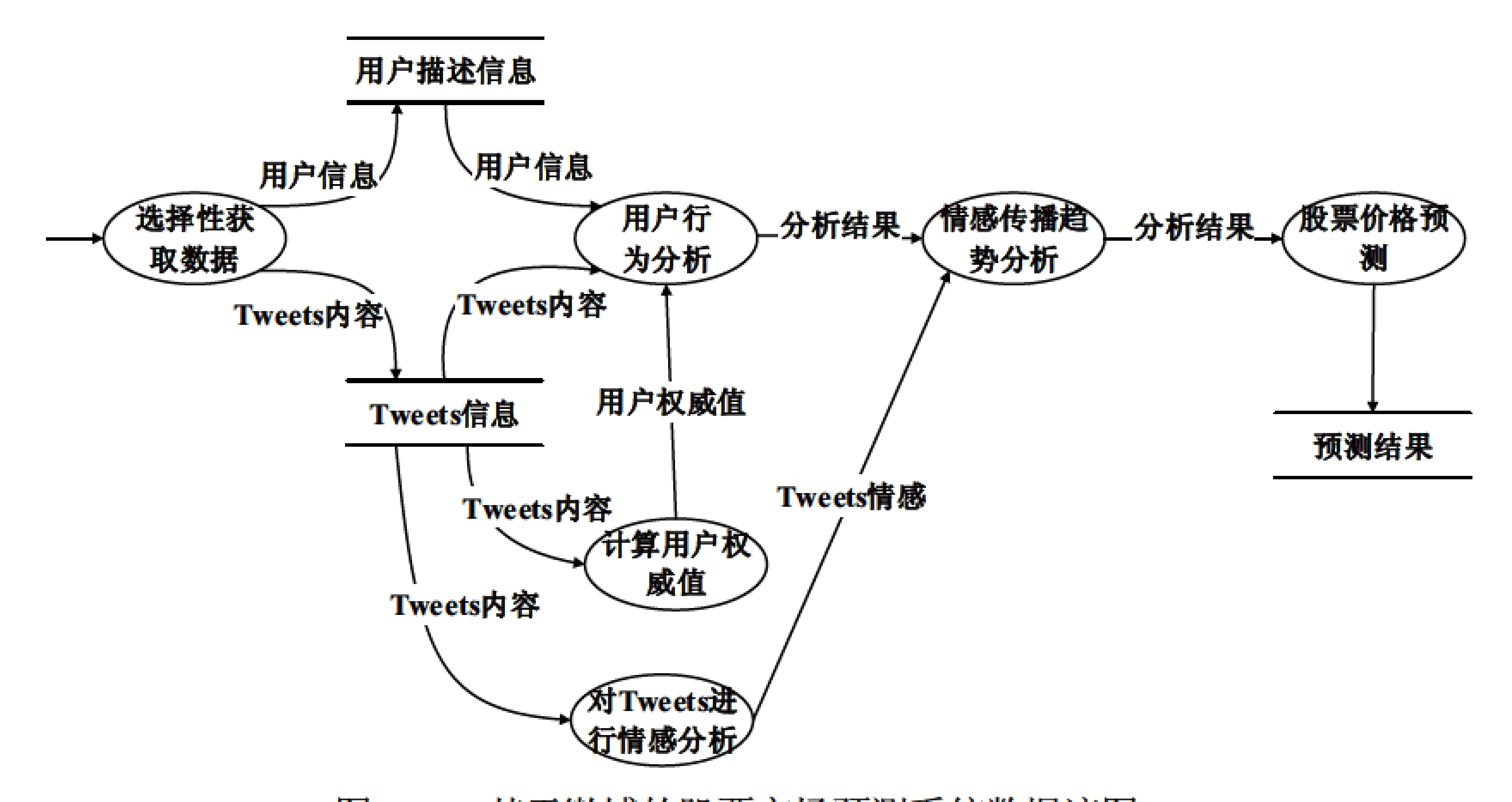


图 10.9　基于微博的股票市场预测系统数据流图

10.3 相关技术介绍

10.3.1　社交网络

1. 社交网络基础

社交网络即社交网络服务（SNS），英文全称为 Social Network Service，是指人和人之间通过朋友、血缘、交易、网络链接、疾病传播、理想、兴趣爱好等关系建立起来的社会网络结构。在网络中，人与人之间通过 Blog、点评、群组等功能，来为网络的用户进行“画像”，当这种“画像”越贴近现实中人的社会性，网络的社会化程度就越高。

社交网络起源于网络社交，随着网络交友的迅速发展，社交网络也在其中慢慢形成、演化、发展，为人们的生活提供更便捷的信息交流。社交网络一直朝着“节约社交时间和物质成本，获取高速、有效的信息”这一方向发展。社交网络通过网络这一平台，把不同的人联系起来，形成具有某一特点的团体。

研究表明，社交网络覆盖了社会的各个层次，上至国家外交，下至家庭关系，并且对于问题的解决，组织的运营，以及个体的成功都起到了决定性作用。

一个社交网络由有限的一群或者多个群的行动个体以及他们之间的相互联系所组成。通常，构成社交网络的主要要素包括个体、群体和关系等。

- 个体：社交网络中的一切个体，表示在一个网络中具体的个人、组织或者是具有其他集体性质的社会实体。通常我们用“节点”来表示每个个体在网络中的位置，任何一个个体都可以形式化为一个节点。
- 群体：把存在某类关系的同一类节点进行一定形式的组合就形成了一个群体，群体之间的成员彼此也可能会有一些相似的关系，是整个网络关系中的一部分。这些所谓的关系可能是同一个地区、学校或者是有相同兴趣爱好等。在群体中总能找到和自己在某些方面存在一定契合度的其他个体，有利于彼此之间信息的交流和分享。
- 关系：指将网络的不同节点相互之间联系起来的属性。通过一定的关系，体现出了社交网络中的个体的社交表现。

社会网络的理论基础源于著名的六度分隔理论和 150 法则。正是基于这两个主要理论，社交网络得到了飞速的发展。

（1）六度分隔理论

六度分隔理论（Six Degrees of Separation），是由美国著名社会心理学家米尔格伦（Stanley Milgram）于 20 世纪 60 年代提出，指的是“你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过六个人你就能够认识任何一个陌生人。

“六度分隔理论”说明了社会中普遍存在的弱纽带发挥着非常强大的作用。有很多人在找工

作时就会体会到这种弱纽带的效果。通过弱纽带，人与人之间的距离变得非常“相近”。Jon Kleinberg 把这个问题变成了一个可以评估的数学模型，我们经常在与新朋友碰面的时候说“世界真小”，因为往往可能大家有共同认识的人。Jon 的研究实证了这个观点。

“六度分隔理论”的发展，使得构建于信息技术与互联网络之上的应用软件越来越人性化、社会化。软件的社会化，即在功能上能够反映和促进真实的社会关系的发展和交往活动的形成，使得人的活动与软件的功能融为一体。“六度分隔理论”的发现和社会性软件的发展向人们表明：社会性软件所构建的“弱链接”，正在人们的生活中扮演越来越重要的作用。

（2）150 法则

150 法则（Rule of 150），是指公认的我们可以与之保持社交关系的人数的最大值是 150。无论你曾经认识多少人，或者通过一种社会性网络服务与多少人建立了弱链接，但是那些强链接仍然符合 150 法则。这也符合“二八”法则，即 80%的社会活动可能被 150 个强链接所占有。

150 法则在现实生活中的应用很广泛。比如中国移动的“动感地带”SIM 卡只能保存 150 个手机号，微软推出的聊天工具“MSN”也只能是一个 MSN 对应 150 个联系人。

2. Twitter 介绍

简单来说，Twitter 是一个免费的社交网络和微型博客服务。任何用户可以发送一条被限制在 140 个字符之内的 Twitter 消息，被称为 tweets，该用户相应的“跟随者”（followers）就能及时查看该信息并发表评论。所谓“跟随者”，就是指关注某一账号所发布内容的其他 Twitter 用户。当然用户也可以主动“追随”（following）别人。它通过限制信息字数、即时抵达、用户自主收发和鉴别真伪的方式，实现了一种自主、互动、简洁、快速的信息传播方式。Twitter 是即时消息的一个变种，它允许用户将自己的最新动态和想法以短消息的形式发送给手机和个性化网站群，而不仅仅发送给个人。它利用无线网络、有线网络、通信技术，进行即时通信，是微博的典型应用。

现如今随着人们对 Twitter 的大量使用，它在商业和金融证券业正在发挥着一定的影响和作用。在企业竞争情报搜集工作中，竞争情报工作者可以利用 Twitter 来实现此前难以完成的功能，一方面可以利用 Twitter 内容发布监控来掌控竞争对手行动信息，另一方面通过 Twitter 交流可视化来构建竞争对手社交网络图。这样就可以使用 Twitter 在竞争情报工作中发挥重要的作用。在金融证券业可以通过 following 一些公司的相关信息和一些证券投资者、基金经理人发布的最新消息，为股票市场的预测和分析提供最新的资讯和信息。

首先，以社交网站的标准来看，Twitter 规模仍然很小，但是 Twitter 可以提供直接的重要的信息，而这一点其他社交网站无法比拟；其次，Twitter 要求用户在该网站上发布文本信息每次不得超过 140 个字。你可以被别人跟帖，别人也可以跟你的贴，这有点类似于 Facebook 的状态更新服务，但它对所有人都是开放的；再次，由于 Twitter 先进的服务，便捷的操作方法，截止到 2011 年初，超过 1.45 亿人都在使用 Twitter，这使得 Twitter 带有了多元的色彩，具有国际因素，这样对于国际上发生的大事能更快、更准确地显示在 Twitter 上，对于不同时间的看法和影响因子得到了大幅度的提升，这样也使我们具有了更多的素材和研究数据。最后，据统计，每天 Twitter 上大约有 25 万与股票相关的微博消息，一些热点股票每天相关的微博数量更是巨大，比如苹果公司股

票$AAPL 以及 Google（谷歌）公司股票$GOOG，每天就有上万条相关微博。正因为如此，针对我们的研究，Twitter 无疑是一个最好的选择。

10.3.2　社交网络表示方法

通常研究者都将网络中的个体抽象化为节点，将个体之间的联系抽象化为连接节点的边，因此大部分的研究工作都是基于图论的研究。随着社交网络的发展，通过对一些具体特性的深入研究，开始有学者将 Hyperbolic Geometry 理论应用到社交网络。

1. 基于图论的表示

一个社交网络用图 G=（V，E，J）表示，V 表示社交网络中的用户集合，是信息内容的载体，E 表示社交网络中用户的关系链接集合。J 表示有向图中的每条边，即用户间的链接所建立的时间。

网络中的每条边与网络中的一对节点是相互对应的。在图论中边分为有向边和无向边以及加权图和无权图，所以根据网络的具体特性，也可以将网络分为以下几类。

- 无向网络：网络中边是无向的，即任意节点对（i, j）和（j, i）对应同一条边。
- 有向网络：节点对（i, j）和（j, i）不属于同一条边。
- 加权网络：网络中的每一条边都有自己相应的权重。
- 无权网络：网络中所有的边都是同等地位的，没有赋予权重值，或者说是网络中所有的边的权重值都为 1。

图 10.10 给出了几种典型的网络结构实例。通常我们所指的网络都默认是指无向无权网络，并且在网络中不存在下面所指两种情况的节点：任何两个节点之间不会有多条边进行连接，即无重边；对某个节点没有连接向自己的边，即无自环。

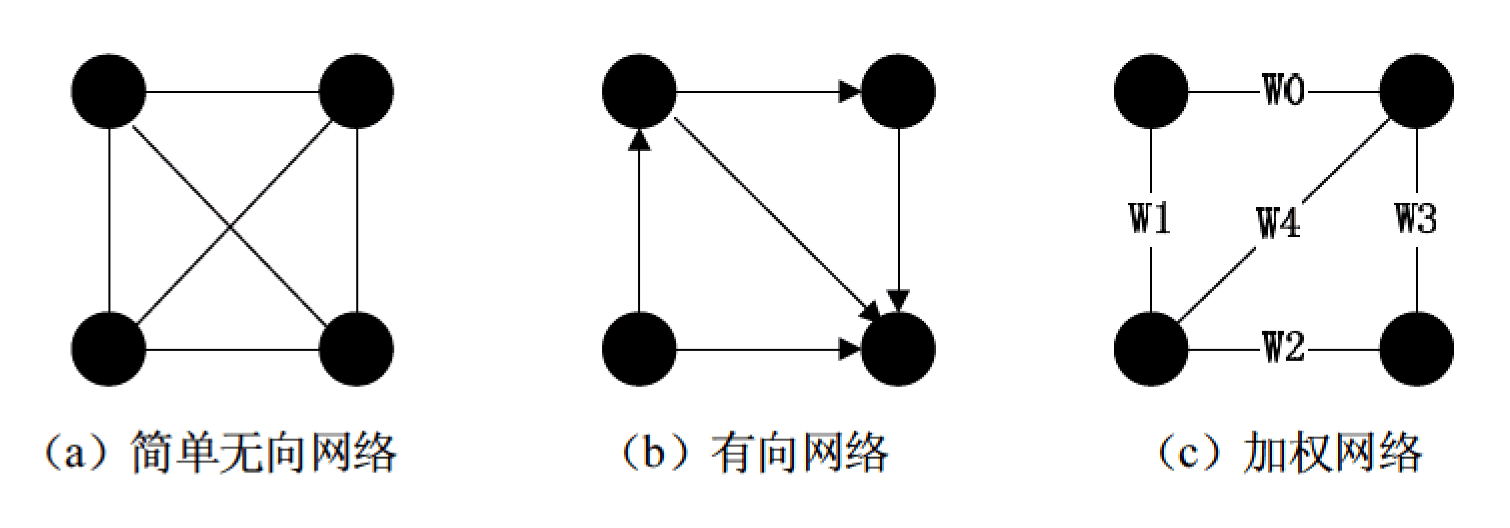


图 10.10　几种网络结构图

微博这种特殊的社交网络用有向图 G=（V，E，J）表示，其中 E 是微博网络中的关系集合，微博中用户 u 关注用户 v 的关系链用（u,v）表示，其中（u,v）∈E，用户 v 关注用户 u 的关系链用（v,u）表示，同样（v,u）∈E。

微博中用户发出的微博信息叫做行为，行为集合用一个表 Actions （User,Action,Time）来表示，Actions 表是由三元组（u,a,t_u）组成，三元组（u,a,t_u）表示用户 u 在时间 t_u 发出行为 a，其中 u∈V。此外，使用 A_u 表示行为集合中用户 u 发出的行为总数，$A_{u\&v}$ 表示行为集合中用户 u 和用

户 v 发出相同行为的数量，而 $A_{u|v}$ 表示行为集合中用户 u 或用户 v 发出的行为数。很明显，有：

$$A_{u|v} = A_u + A_v - A_{u\&v} \qquad \text{式（10-1）}$$

A_{v2u} 表示行为集合中用户 v 发出的行为传播到用户 u 的数量，即用户 v 影响用户 u 的行为总数。

2. 基于矩阵的表示

（1）邻接矩阵

邻接矩阵是指矩阵中的元素 a_{ij} 是从节点 i 到节点 j 的边的条数。如果一个网络中有 n（i 节点表示为 n_i）个节点，那么邻接矩阵 A=（a_{ij}）是一个 n*n 的矩阵，其中若 n_i 邻接 n_j，那么 a_{ij}=1，否则 a_{ij}=0。如果为加权网络，则只需要矩阵不为 0 的边信息存放相应边的权值即可。简单网络关系拓扑图及其邻接矩阵表示如图 10.11 所示。

（2）关联矩阵

关联矩阵是关于网络中节点和关联边之间关系的矩阵，记为 R=（r_{ij}）。对于一个图 G，记 p 为节点个数，q 为边的总数，r_{ij} 表示在关联矩阵中节点 i 和边 j 之间的关系，若节点 i 和边 j 之间是连着的，则 r_{ij} =1，否则 r_{ij} =0。如果为加权网络，则只需要矩阵不为 0 的边信息存放相应边的权值即可。简单网络拓扑图及其关联矩阵表示如图 10.12 所示。

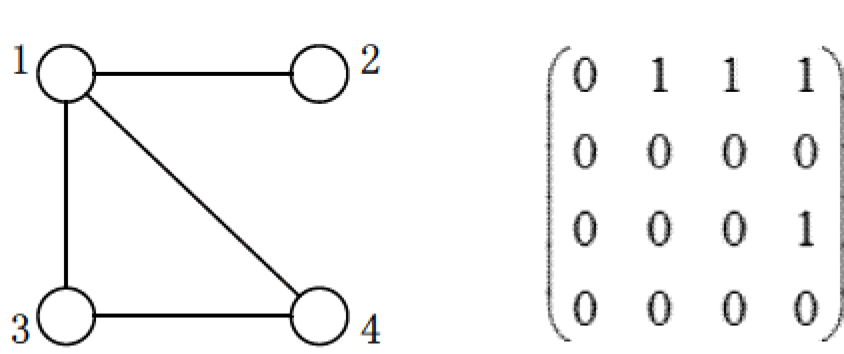
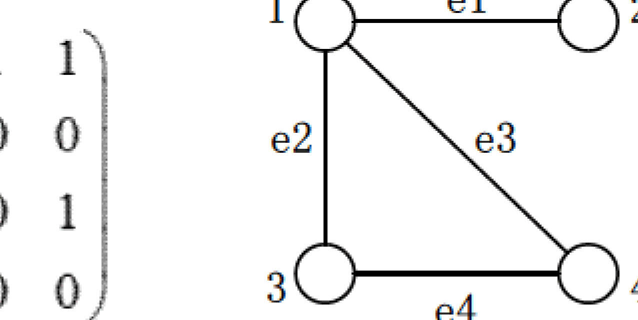


图 10.11　简单网络拓扑图及邻接矩阵表示



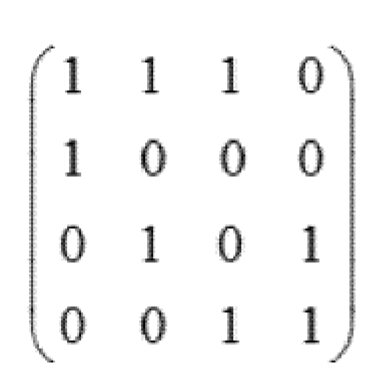


图 10.12　简单网络拓扑图及关联矩阵表示

10.3.3　信息传播模型

任何传播模型都包含两个因素：传播规则和网络拓扑结构。研究网络上的传播行为，通常不区分其对象是计算机病毒还是在人群中传播的疾病，通常能在网络上传播开的东西在研究时都被叫做传染病，哪怕它只是一个谣言。目前研究最广泛的传染病模型是 SIR 和 SIS 模型。这两种模型实际上都规定了传播规则，但都是无结构的。此处无结构是指任何两个个体之间都有联系，换言之，传播可以在任何两个个体之间发生，或者说任何两个个体之间有着同样的传播行为。后来的包括研究信息传播或舆论形成的模型都是有结构的传播模型。

1. 无结构传播模型

（1）SIR 模型

基本状态包括：易感状态（S）、感染状态（I）和移除状态（R）。易感状态是健康状态，但有可能被感染；感染状态即染病的状态，其具有传染性；移除状态即感染后被治愈并获得了免疫

力或感染后死亡的状态，其不具有传染性，也不会被传染。他们不再会对其他人产生影响，所以可以看作已经从系统中移除。易感个体被感染，然后恢复健康并获得免疫力，这样的传播模型我们称为 SIR 模型。

假设易感染个体在单位时间内被某个染病个体传染的概率为 β，而染病个体康复的概率为 γ，并用 s、i、r 分别标记 S、I、R 类个体所占比例，则在 SIR 模型中，传染病传播可以用下面的微分方程组来描述。

$$\frac{ds}{dt}=-\beta is,\frac{di}{dt}=\beta is-\gamma i,\frac{dr}{dt}=\gamma i$$ 式（10-2）

（2）SIS 模型

SIS 传播模型中的个体只存在两种状态：易感状态（S）和感染状态（1）。易感个体被感染，然后恢复健康，但其有可能再次被传染，所以是从感染状态恢复到易感状态，这样的传播模型我们称为 SIS 模型。假设易感个体和感染个体的在总体中所占的比例分别为 s 和 i。在每个单位时间内，如果系统中易感个体被感染的概率为 β；同时，感染个体被治愈，变为易感个体的概率为 γ。SIS 模型对应的微分方程描述为：

$$\frac{ds}{dt}=-\beta is+\gamma i,\frac{di}{dt}=\beta is-\gamma i$$ 式（10-3）

2. 有结构传播模型

以上两种传播模型并没有考虑人际网络的结构，是在无结构的群体上的传播模型。当然也可以把这种无结构群体看成是有结构的，即它是一个完全图。也就是说，在这个图上，任何两个人之间都有联系。但这在现实生活中是不可能的。只有在一个非常小的群体上才有可能实现任何两个人之间都有联系。由前面所提到的 150 法则可知，这样的群体的数目的上限是 151。当然实际中的这种群体一定是远远低于这个上限的。比如学校中的两个班级，班内的每个同学之间都有联系是可能的，但要求两个班中任何两个同学都相互认识并不容易。所以实际信息传播的人际网络是有结构的。基于这种情况，提出的传播模型主要有社会影响模型。

社会影响模型的主要特点是：

- 每个个体的观点用 σ 表示，每个个体可以取两种观点，即 $\sigma\in\{-1,+1\}$；
- 每个个体受到群体中所有其他个体的影响，其中，与该个体观点相同的个体会给他支持，而与该个体观点相反的个体会给他劝说。即，若 j 与某个体观点相同，则 j 会给该个体支持，其支持程度的大小用 S_j 表示，而 j 对与其观点相反的个体的劝说程度大小用 P_j 表示；
- 社会群体对其中个体施加的影响力是群体中个体的数目及他们的社会直接作用力的增函数；
- 人与人之间作用力的大小与其之间的“社会距离”成反比；

系统的演化规则如下：

$$\sigma_i(t+1) = -sign(\sigma_i(t)I_i(t)+h_i)$$ 式（10-4）

其中，I_i是系统中所有人对 i 的社会影响力总和，h 是随机变量，引入系统噪声。这个模型类似于无结构的。因为根据定义，任何个体之间都有影响。但这个模型又与前面的模型不同，其中任何人的影响力是不一样的，其大小与“社会距离”有关。而前面模型中，所有人的影响力是一样的。

上面提出有结构传播模型中，人际关系网的结构是规则结构，它定义在二维方格上。但实际的人际关系网络却不是一个规则的网络，很显然，不同人的朋友的数目不可能完全一样，这表现在人际网络上就是节点的度是有变化的。所以研究信息传播模型的时候，不仅要给出合理的传播规则，还要给出合理的网络结构。

10.4 详细设计与实现

10.4.1 Twitter 数据采集模块详细设计

Twitter 为了鼓励第三方应用软件的发展和应用，以 API（Application Programming Interface）的方式开放了一些应用接口，这种应用接口也叫开放 API。网站提供开放平台的 API 后，可以吸引一些第三方的开发人员在该平台上开发商业应用，平台提供商可以获得更多的流量与市场份额。而通过 Twitter 开放的 API，任何对 Twitter 平台有兴趣的开发者都可以通过 API 开发基于 Twitter 平台的软件应用，来优化产品对 Twitter 用户的体验。

如今，Twitter 提供两种 API，其中一种是 Twitter 官方开发的，用于获取用户信息、用户微博、用户认证等功能，另外一种则是 Twitter 收购 Summize 而得到的 search API，提供与 Twitter 相关的搜索，尤其是实时搜索。今天的 Twitter API 系统每天收到超过三十亿次的请求，这里需要的股票言论相关的微博则通过 Twitter 的搜索 API 和用户信息 API 获得。

Twitter 平台大多数 API 请求都要求用户认证，返回的结果均与认证用户有关系。最简单的认证方法是 HTTP Basic Authentication 验证机制，使用在 Twitter 中注册用户名作为 Session 或 Cookie 的用户名部分内容。Twitter API 根据用户特定的请求返回对应特定格式的数据，平台目前支持以下 4 种数据返回格式：XML、JSON、RSS、Atom。开发者可以在每次请求时使用不同的请求方法指定不同的返回结果。本系统在提取股票言论相关的微博内容时，采用 XML 的返回格式，根据设计的 XML 解析工具对内容进行解析，并将返回内容存入到数据库中。截止到编写时间，Twitter API 已更新到 1.1 版本，与之前版本相比较，API 在资源访问地址以及参数列表方面都有较大不同，开发者应该根据实际情况选择合适的 API 版本。以下是提取微博内容主要用到的 Twitter API 介绍，它基于最新 1.1 版本。

- 返回能够匹配查询字符串的相关微博集合，GET search/tweets。资源访问地址为 https://api.twitter.com/1.1/search/tweets.xml，API 接受的部分参数如表 10.10 所示。

表 10.10　GET search/tweets API 的参数列表

参数	参数描述
q（required）	查询字符串，最长 1000 个字符
count	返回每页的最多结果数量，最大值 100，默认值 15
return_type	指定返回结果的类型 可选类型：mixed，recent，popular
since_id	返回结果中的微博 ID 比指定值大
max_id	返回结果中的微博 ID 小于或等于指定值

- 返回指定用户的扩展信息，需要给定用户的 id 或显示名称，GET users/show。扩展信息包括用户的页面设置、微博数量等，并且用户最近发表的微博消息也可能包括在内。资源访问地址为 http://api.twitter.com/1.1/users/show.xml，当访问 Twitter 用户 rsarver 时，访问以下地址：

```
https://api.twitter.com/1.1/users/show.xml?screen_name=rsarver
```

- 返回指定用户在 Twitter 中跟随（follow）的用户，在 API 中以 friends 表示，GET friends/ids。返回列表按照指定用户最近跟随的用户优先的顺序排列，资源访问地址为 https://api.twitter.com/1.1/friends/ids.xml，当提取用户 twitter API 关注的 Twitter 用户时，访问：

```
https://api.twitter.com/1.1/friends/ids.xml?screen_name=twitterapi
```

- 返回指定用户在 Twitter 中被跟随的用户（followers），GET followers/ids。资源访问地址为 https://api.twitter.com/1.1/followers/ids.xml，当提取用户的粉丝时，访问：

```
https://api.twitter.com/1.1/followers/ids.xml?screen_name=sitestreams
```

这里通过 Twitter 提供的 API，按照上文中的使用形式，从 Twitter 平台中提取需要的股票言论相关的微博内容。比如，按照关键字“$AAPL”搜索苹果公司股票相关的微博消息，搜索结果中，id 为 29242868 的用户于 2011-08-09 18:32:36 发布一条微博“$aapl get in now good news 3rd qtr great and 4 qtr even better，300% increase in rev in last 90 day”，然后根据该用户 id 通过 API GET users/show，查询到用户名为 TrendRida，使用语言为 eng，关注 FinanceTrends 等 188 个用户，粉丝有 Mark_Lexus 等 2427 个用户，最终将提取的数据整理并存入数据库中。

值得注意的是，Twitter 平台 API 在使用上是有限制的，每个客户端每小时最多发送 150 次请求，每次请求返回微博内容的发布时间不超过一周。本文在使用 Twitter API 提取股票言论相关微博时，需要定时发送请求，以免超过时间或内容限制，造成程序不必要的影响。另外，由于众所周知的原因，Twitter 内容在国内无法访问，这里使用 GoAgent 代理访问 Twitter。

根据系统的需求，需要从 Twitter 中获取两种信息：用户信息和 Tweets 信息。相关的类结构和流程图如下。

- 用户（User）类：包括用户 ID（id），用户名字（name），用户昵称（screen_name），用户位置（location），语言（lang），Tweets 内容（status，status 的数据类型是 Status 类）。

类图如图 10.13 所示。

- Tweets 信息（Status）类：包括创建时间（created_at），Tweet ID（id），Tweet 文本内容（text），Tweet 来源（source），回复该条 Tweet 的用户 ID（in_reply_to_status_id），回复 Tweet 的 Tweet ID（in_reply_to_user_id），回复 Tweet 的用户昵称（in_reply_to_screen_name）。其类图如图 10.14 所示。

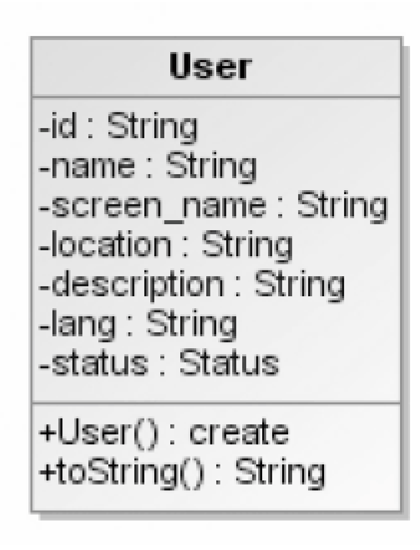


图 10.13 用户类图

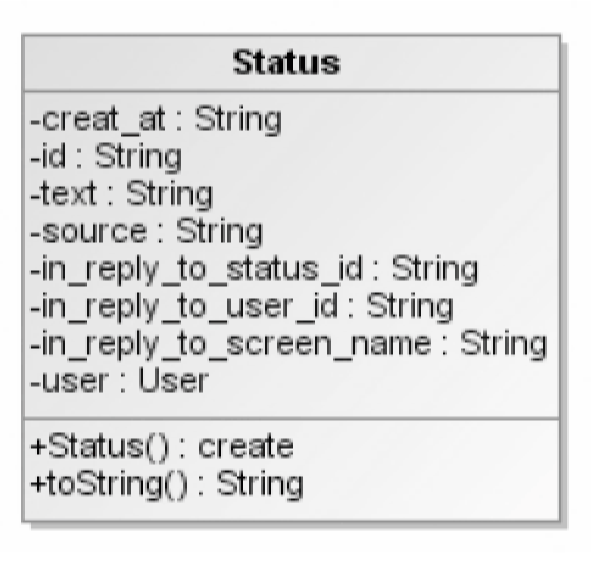


图 10.14 消息类图

- Tweets 数据提取流程

根据图 10.15 所示，获取到的数据通过解析类 ConvertBean 对获取的 XML 进行解析，该解析类的类图如图 10.16 所示。随后通过 Receiver 类返回最终获取的数据，且该类类图如图 10.17 所示：

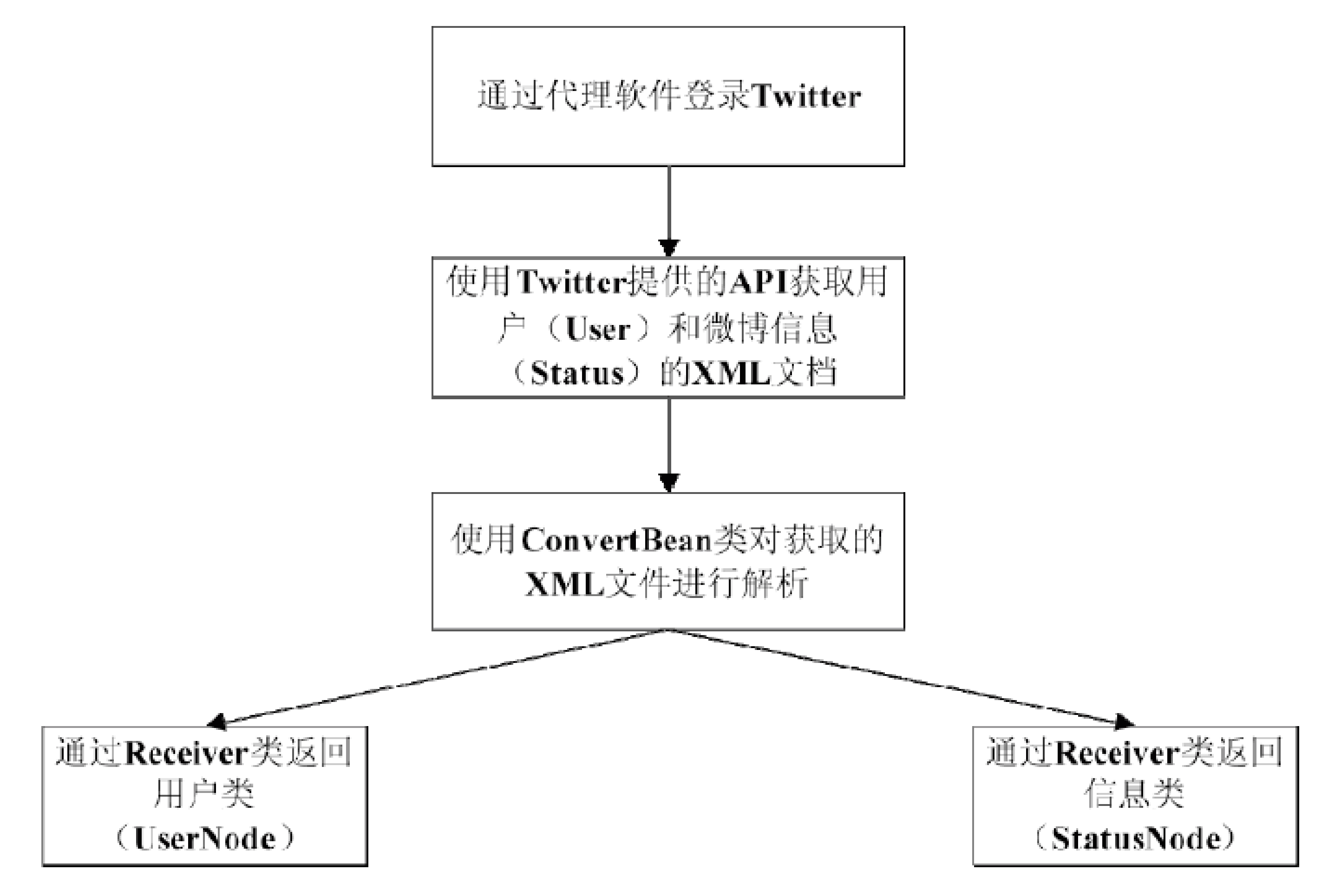


图 10.15 Twitter 数据采集流程

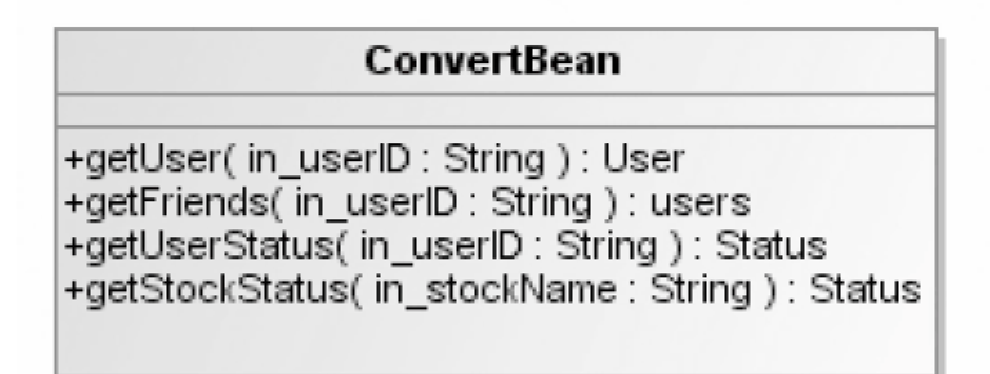


图 10.16 ConvertBean 类图

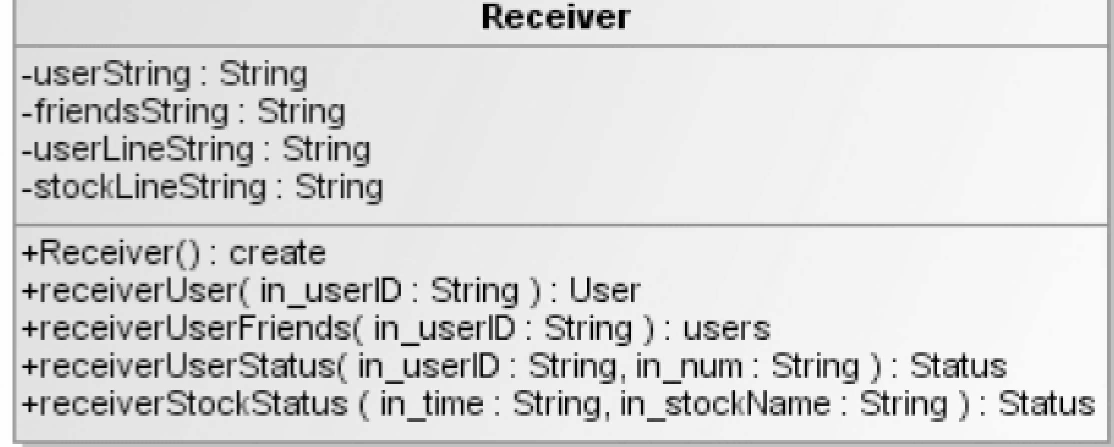


图 10.17 Receiver 类图

10.4.2　Twitter 数据分析模块详细设计

本节主要介绍 Twitter 数据分析模块的设计，包括股票微博涨跌情感分析和 Twitter 平台用户权威值计算两部分。

1. 股票微博涨跌情感分析

本系统在分析股票言论的看涨或看跌情感在微博网络中传播之前，先要对股票言论微博进行情感分析。对于股票言论而言，股票相关的情感有看涨、看跌和持平（波动）三种情感。由于波动情感在判断上的复杂度，为了简单起见，这里只分析股票微博的看涨和看跌情感。

Twitter 使得互联网用户可以随时随地的接入到网络，并随心所欲地发表自己的观点和情感，这导致 Twitter 上的微博内容杂乱无章，以网络语言为主，无固定结构，无规律可循。针对 Twitter 微博内容的短文本特征和噪声特征，本文提出的股票微博的情感分析分为 5 个步骤：文本规范化、分词、去除停止词、绑定否定词和匹配情感字典。具体流程如图 10.18 所示。

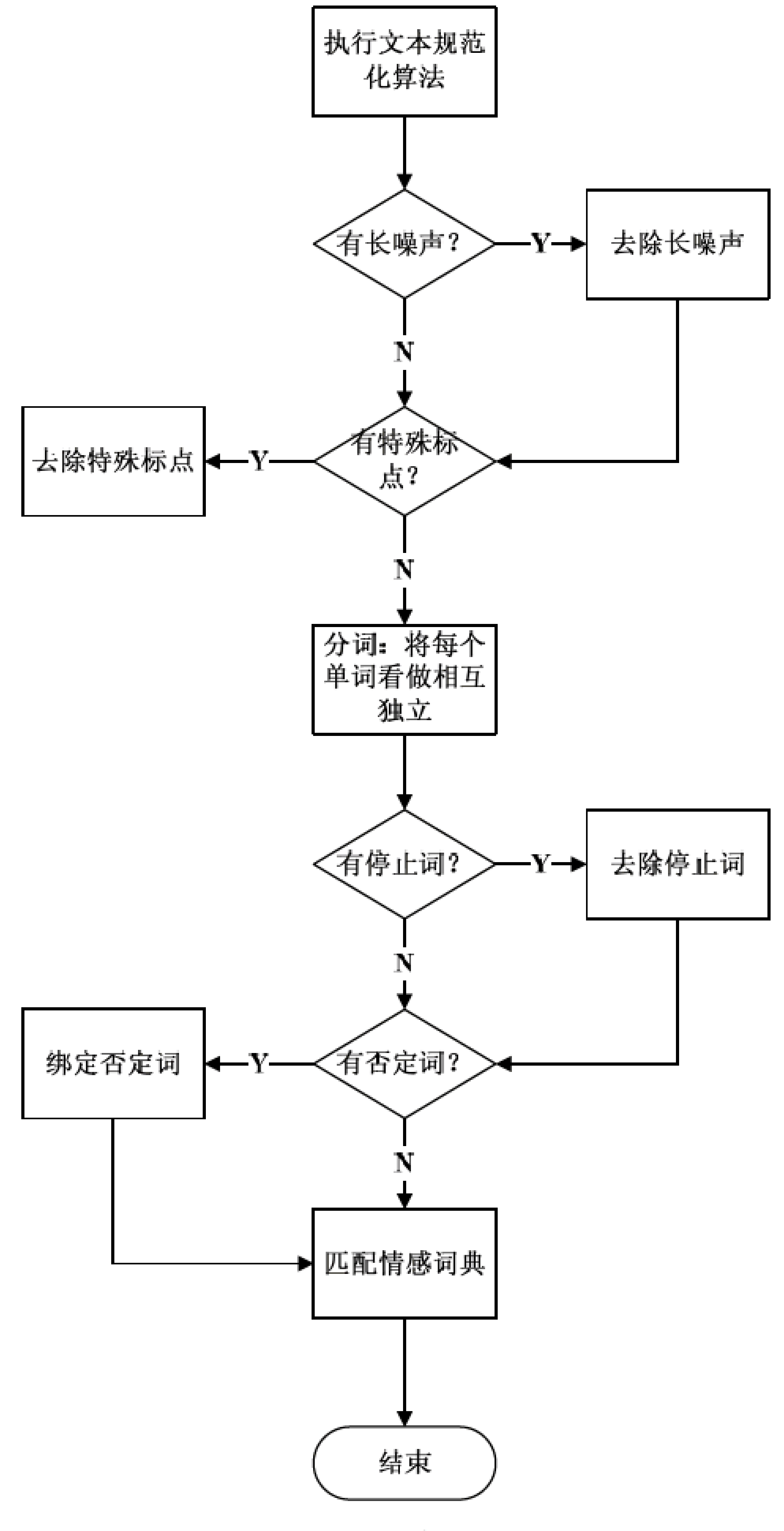


图 10.18　情感分析流程

以微博内容“RT @Reuters: Apple briefly edged past Exxon Mobil to become the most valuable company in the U.S. http://t.co/vOtnNjD $AAPL”为例，情感分析的流程图如图 10.19 所示。

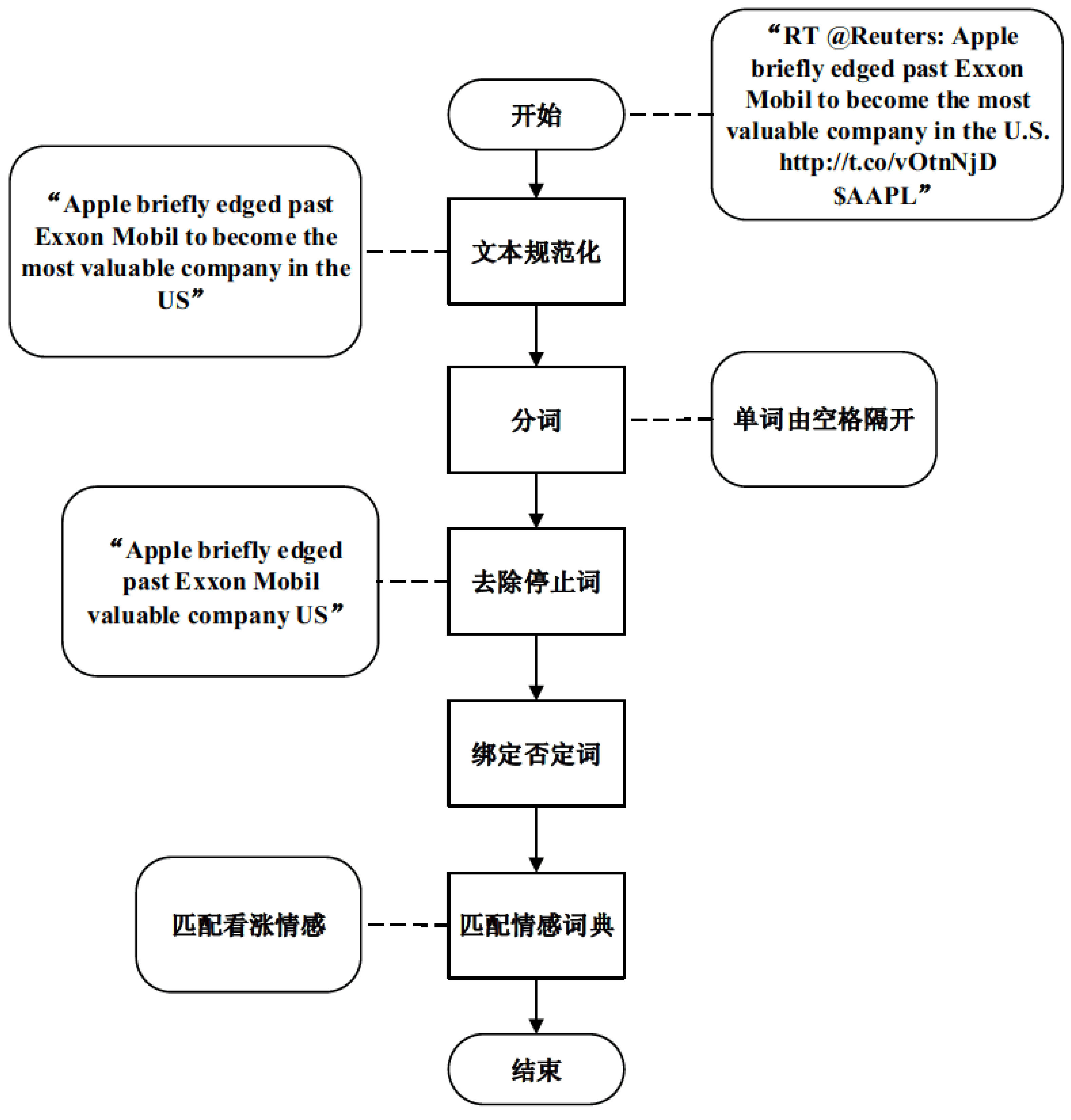


图 10.19　股票微博涨跌情感判断流程图

（1）文本规范化

Twitter 微博内容存在网页链接、用户对话、标点符号、表情符号、专用词汇等噪声，在分析股票言论情感时，先对微博内容进行文本去噪，将微博内容规范化，只保留股票言论相关内容。

对“RT @Reuters: Apple briefly edged past Exxon Mobil to become the most valuable company in the U.S. http://t.co/vOtnNjD $AAPL”微博进行规范化，微博中“http://t.co/vOtnNjD”属于网络链接，“RT”是 Twitter 专用词汇——转推，“@Reuters”是用户对话，这些都属于非规范化内容，将原微博文本规范化后的结果是“Apple briefly edged past Exxon Mobil to become the most valuable company in the US”。

（2）分词

由于情感分析是对文本中有意义的单词或词组进行计算，所以必须要对微博文本内容进行分词处理。Twitter 的每条微博内容局限在 140 个英文字符内，每个单词的信息量很大，因此本文使用的分词方法就是简单的以空格符和标点符号为边界，并且只考虑英文部分。本文的分词方法虽然简单，但是对于 Twitter 微博文本来说却是很适用的。

（3）去除停止词

停止词，是由英文单词 stopword 翻译过来的，通常意义上，停用词大致分为两类，一类是人类语言中包含的功能词，这些功能词极其普遍但没有什么实际含义，比如 the、is、at、which、on 等。另一类包括词汇词，比如 want 等。这些词的应用十分广泛，但是文本分析无法保证能够给出真正相关的结果，会降低处理的效率，所以通常会把这些词从文本中移去，从而提高文本分析性能。

这里使用的停止词都是人工输入、非自动化生成的，人工输入的停止词形成一个停止词表，确保停止词的过滤符合股票言论情感分析的需求。对于文本规范化及分词后的微博"Apple briefly edged past Exxon Mobil to become the most valuable company in the US"，去除其中的停止词 to、become、most、in 和 the，最终结果为"Apple briefly edged past Exxon Mobil valuable company US"。

（4）绑定否定词

英语等语言中表示否定意义的词，有独特的语法意义和影响，如英语中的 no、not、never、hardly、rarely、few、little、seldom 等。本文将否定词出现位置的前一个词和后一个词组成两种组合，例如 I do not like $GOOG，将被视为 do+not 和 not+like 两种。

（5）匹配情感字典

情感词指蕴含情感色彩的词语，它可以表达用户的内心情绪。情感色彩是用户对客观事物所持有的态度，通常分为褒义和贬义两种情感色彩。文本情感分析中，主要通过对句子中情感词的识别，作为判断文本情感倾向的依据。对于股票言论相关的微博内容来说，看涨情感和看跌情感分别为用户对股票持有的两种截然相反的态度。为分析微博内容的情感倾向，这里建立了看涨情感字典以及看跌情感字典。经过前面 4 个步骤，处理后的微博文本将和两种词典逐一比对，记录匹配程度，进而得到用户的情感：看涨或者看跌。本节采用的例子中存在 valuable 一词，而该单词就是看涨情感字典中判断用户看涨情感有力的依据，所以得到该用户对苹果公司股票$AAPL 持有看涨的态度。

2. Twitter 平台用户权威值计算

在计算用户之间的行为影响概率之前，首先要计算社交网络中每个用户的主题权威值。主题就是用户的兴趣点所在，对于股票市场来说，用户关注的主题就是特定的某支股票，比如 AAPL（苹果股票）。用户权威值不同，则用户对社交网络中其他用户的影响就必然不同。本小节主要介绍微博在线社交网络中针对股票主题的用户权威值的计算方法。

Twitter 是世界范围内用户量最多、知名度最广的在线社交网络，Twitter 微博与其他国内外的微博平台相比，具有相同的基本功能，但是也具有其独特的特征，Twitter 平台的每一条微博称为 tweet。这里针对社交网络的所有研究都是基于 Twitter 平台微博的。

首先，先将 Twitter 中的微博分为三类：原创微博（Original Tweet，OT）、对话微博（Conversational Tweet，CT）、转发微博（Re Tweet，RT）。这三类微博的定义及区别如下：

- 原创微博（OT）：由 Twitter 用户发表的非对话微博和转发微博的微博内容。
- 对话微博（CT）：Twitter 用户直接对另一个用户发表的微博，表现在微博内容中就是微

博最前添加了对方的 Twitter 用户名（@username）。用户发表对话微博可以通过在对方微博下单击“回复”按钮。

- 转发微博（RT）：Twitter 用户直接复制另一个用户的微博，发表在自己的微博列表中，从而在用户自己的网络中传播。

三类微博在 Twitter 平台的表现形式如图 10.20 所示，以 Twitter 用户@TwitterOSS 为例：



图 10.20　三类微博的表现形式

其次，根据以上三种分类，提出 Twitter 平台中用户所发历史微博的几种统计值，如表 10.11 所示。

表 10.11　主题用户的统计值

统计值	统计值介绍
OT_1	原创微博的数量
CT_1	对话微博的数量
CT_2	由 Twitter 用户自身发起的对话微博的数量
RT_1	转发其他用户微博的数量
RT_2	被其他用户转发的微博数量
RT_3	转发自身微博的用户数量

经过扫描微博数据库中的用户历史微博，可以统计出表 10.11 中每一项属性。然后，结合这些属性并针对每一个用户，创建了微博用户的 5 种特征，分别从不同角度描述了用户的影响力。

（1）主题参与度（Topic Involved）

针对每一个指定用户，从微博数据库中的历史微博中分析该用户关于某特定主题的参与程度。

$$\text{主题参与度(TI)} = \frac{|\#topic|}{OT_1 + CT_1 + RT_1} \qquad \text{式（10-5）}$$

式（10-5）中，|#topic|表示用户所有历史微博中存在#topic 样式的哈希标签的微博数量。简单地说，哈希标签就是微博搜索时的关键字，在 Twitter 中的书写形式是将关键字单词放在“#”之后，因为英文单词之间以空格分隔，Twitter 中“#”后面只能跟一个关键词。主题参与度 TI 表明了该用户对

于某个主题的参与程度，如果以股票为例，上式中的#topic 可以是$AAPL，$标签是 Twitter 在 2012 年推出的新标签，用于关联股票代码，此时 TI 表明了该用户投入到苹果公司股票的微博讨论的程度。

（2）原创度（Original Strength）

原创度表明了用户对于某主题独立思考的程度，该特征也从另一个侧面反映了用户在某主题上的权威值。如式（10-6）所示。

$$原创度(\mathrm{OS})=\frac{OT_1}{OT_1+RT_1} \qquad 式（10-6）$$

在某主题上比较有权威的用户自然不会追随其他用户的言论和想法，一般情况下，权威用户一旦发表了自己的独立看法，立即会极大影响到所有子节点用户，并快速地在自己的网络中开始传播。所以，微博用户的原创度表明了用户在某一主题上的影响力。对于真正具有极大权威的用户，其原创度一定趋向于 1。

（3）非对话强度（Non-Conversation Strength）

非对话强度标示了一个用户真正切实参与到主题的程度，也侧面标示了用户偏离主题而进入和其他用户聊天状态的程度。非对话强度可以用公式 $OT_1/（OT_1+CT_1）$表示，但是非对话强度的这种表示明显忽略了以下这种情形：某些用户虽然发出对话类微博，但并不是其主动发出，而是因为出于礼貌而对其他用户的微博进行回复。在这种情况下，该用户的实际非对话强度应该会比上式的计算结果有所增大，因此对上面的非对话强度公式进行了修正，如式（10-7）所示。

$$非对话强度(\overline{C}\mathrm{S})=\frac{OT_1}{OT_1+CT_1}+\lambda\frac{CT_1-CT_2}{CT_1+1} \qquad 式（10-7）$$

公式修正部分的加入对于计算个人用户的权威值更有意义，因为个人用户与机构组织用户不同，个人用户更具有社交性，更容易和其他微博用户产生沟通交流，因此上式对于个人用户的权威值计算更具有精确性。

非对话强度$\overline{C}\mathrm{S}$的值需要满足$\overline{C}\mathrm{S}<OT_l/(OT_1+CT_2)$，将该约束条件代入非对话强度公式可得到：

$$\lambda<\frac{OT_1*(CT_1+1)}{(OT_1+CT_1)*(OT_1+CT_2)} \qquad 式（10-8）$$

根据对数据库中微博数据的计算，本系统认为$\lambda\approx0.05$时，非对话强度$\overline{C}\mathrm{S}$的计算值满足前面提到的约束条件。λ取值过高可导致个人用户以及活跃用户的权威值增大，反之则会导致机构组织用户的权值过大。

（4）内容影响力

在线社交网络中用户的微博内容影响力通过该用户被网络中其他用户转发的微博数量来计算，具体计算公式如式（10-9）所示：

$$内容影响力(\mathrm{CI})=RT_2*\log(RT_3) \qquad 式（10-9）$$

内容影响力表明了用户所发出的微博内容在整个社交网络中所造成的影响程度。由于内容影响力的计算通过该用户被其他用户转发的微博数量来衡量，因此必须考虑网络中其他某用户过于信任或欣赏该用户，导致只有少数几个人转发该用户大量的微博。为了抑制这种情况对内容影响

力计算公式产生的副作用，内容影响力公式引入了 RT_3。

对数函数 log(n)的定义域是 n>0，但是在内容影响力计算公式中，RT3 可能等于 0，RT3=0 表示转发自己微博的用户数量为 0。当这种情况发生时，该用户被转发的微博数量肯定也为 0，即 RT3=0 RT2=0。因此这里规定，当 RT3=0 时，CI=0。此时，内容影响力 CI 的计算公式覆盖了微博社交网络中的所有情况。

（5）内容相似度

内容相似度反映了用户最近发布的微博内容与之前发布的微博内容的相似程度。两个单词集合 S_1, S_2 的相似度 $S(S_1, S_2)$ 使用公式 10-10 计算：

$$S(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1|} \qquad \text{式（10-10）}$$

值得注意的是，公式 $S(S_1, S_2)$ 不具有对称性，即 $S(S_1, S_2) \neq S(S_1, S_2)$。得到任意两个单词集合的相似度 $S(S_1, S_2)$ 后，就可以对某个用户所发表的所有微博内容进行相似度计算，计算方式是对该用户历史发表的所有微博内容的相似度取平均值，公式如式（10-11）所示。

$$SS = \frac{2 * \sum_{i=1}^{n} \sum_{j=1}^{i-1} S(s_i, s_j)}{n * (n-1)} \qquad \text{式（10-11）}$$

在式（10-11）中，假设用户在历史共发布 n 条微博，并且 n 条微博是按时间排序的，即∀i<j，都有 time（S_i）<time（S_1）。

内容相似度 SS 值较高说明该用户关注的主题比较集中，因此他在网络中的权威值就比较高，而内容相似度比较低的用户，关注了很多不同的主题，具有广泛的兴趣，在某一主题上的权威值会有所降低。

综上所述，作者从 5 个方面介绍了可标识微博用户针对某主题的权威值的特征：主题参与度 TI、原创度 OS、非对话强度 $\overline{C}S$、内容影响力 CI 以及内容相似度 SS。每种特征都可以从数值上体现该用户在某主题上的权威大小。通过将这些特征值求和，就可以对 Twitter 在线社交网络中的用户按照某主题进行权威值排序，如图 10.21 所示。

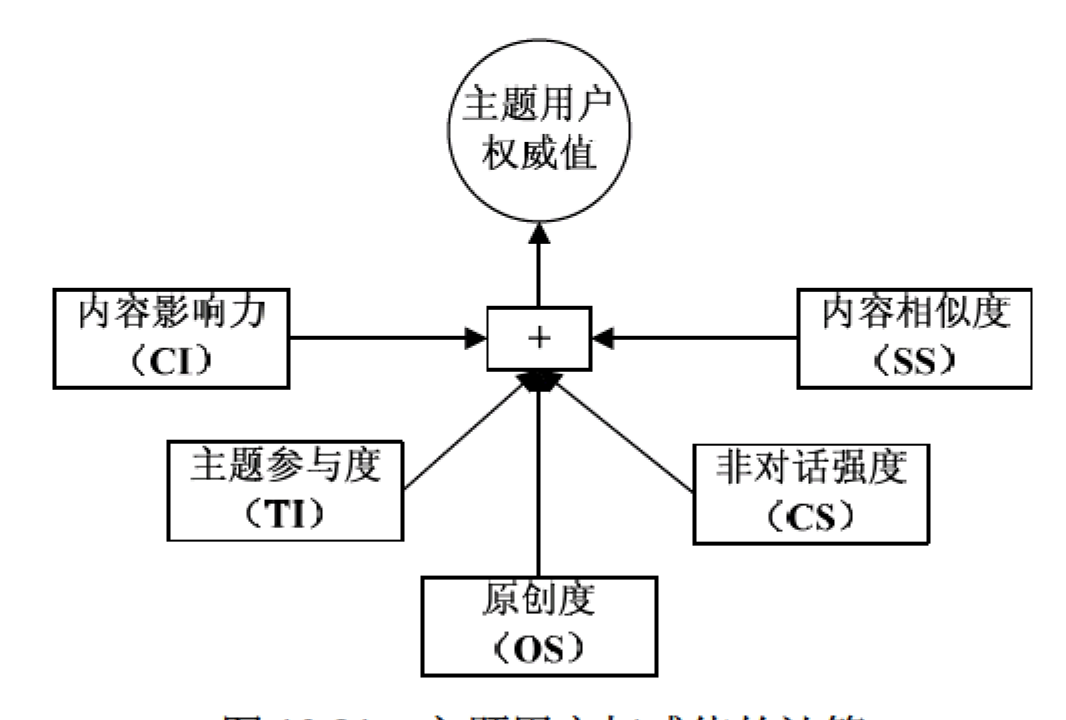


图 10.21　主题用户权威值的计算

根据前文介绍，计算用户权威值时，需要统计出三类微博的数量，可以通过遍历用户发出的

历史微博进行数量统计。原创微博 OT_1、对话微博 CT_1 与 OT_2、转发微博 RT_1 通过 10.4.1 节介绍的内容在文本特征上加以区分和统计，而转发微博 RT_2 和 RT_3 的计算则需要对微博具体的属性进行分析。Twitter 平台每条微博均有转发属性：是否被转发、转发用户数。例如 Twitter 用户 twitterapi 在 2012 年 6 月 6 日发布的一条微博"Along with our new #Twitterbird, we've also updated our is play Guidelines: https://t.co/Ed4omjYsJC"，在其 XML 格式中有以下几种属性：

```
{"text": "Along with our new #Twitterbird, we've also updated our Display Guidelines: https://t.co/Ed4omjYsJC",
 "retweeted": true
 "retweet_count": 66
 "screen_name": "twitterapi"}
```

通过 retweeted 属性可在遍历时统计被其他用户转发的微博数量 RT_2，通过 retweet_count 属性可以统计总共转发自己微博的用户数量 RT_3。

根据用户三类微博的各个属性值，很容易通过公式（10-5）~公式（10-11）计算微博用户的主题参与度、原创度、非对话强度、内容影响力以及内容相似度的数值。值得注意的是，在用户主题参与度计算中，如果计算用户对于苹果股票的参与度，就需要提取历史微博中所有提及 $AAPL 的微博数量作为|#topic|的值。

在计算用户微博内容相似度时，首先要去除文本噪声，即从用户微博内容中去除标点符号、停止词等内容，将每条微博内容转变为单词集合，这样处理可使内容相似度的计算更具鲁棒性。一般的微博在线社交网络对于微博内容都有字数限制，Twitter 限制每条微博内容不得超过 140 个英文字符，微博的字数限制决定了文本去噪后的每个单词集合不可能会无限膨胀，内容相似度的计算变得简单。

通过公式 $W(u) = TI + OS + \overline{C}S + CI + SS$ 对上文中计算出的 5 种特征值进行加和，便可求出 Twitter 平台上每个用户的权威值 W(u)。

10.4.3　用户行为分析模块详细设计

本小节提出用户行为影响概率模型的模型框架——一般阈值模型。微博中用户被信息影响或者称该用户被激活，就会成为传播节点，将该行为在社交网络中继续传播下去。某一时刻未被影响的用户节点 u 周围已经有若干父传播节点，它们形成对用户 u 的影响用户集合 S，集合中任意用户节点 $v \in S$ 都是在用户 u 关注用户 v 之后被激活的。影响用户集合 S 中的任意用户 v 均会以一定的概率激活用户 u，从而集合 S 中的所有用户会形成影响联合概率 P_u（S）而对用户 u 产生影响，用户 u 被影响后就会发出同样的行为。

本小结主要通过计算影响用户集合 S 对用户 u 的影响联合概率，对单个用户 u 的发出行为做出预测。一般阈值模型中的阈值 θ_u 指用户 u 的受影响阈值，当 $P_u(S) \geq \theta_u$ 时，可以预测用户 u 将会被父节点影响，从而成为传播节点，获得传染性，可以将行为继续传播到子节点中未被影响的用户节点。

根据微博中用户间的实际影响关系，显然可知，影响联合概率函数 $P_u(S)$ 是单调的，如果 $S \subseteq T$，一定有 $P_u(S) \leqslant P_u(T)$。而且用户 u 的所有父节点之间具有比较弱的联系，父节点对用户 u 的影响概率可以看作是独立的。

因此，影响联合概率可以被定义为：

$$P_u(S) = 1 - \Pi_{ves}(1 - P_{v,u}) \qquad \text{式（10-12）}$$

公式（10-12）中，$P_{v,u}$ 指用户 v 对用户 u 的行为影响概率，也就是行为从用户 v 传播到用户 u 的概率。行为传播有延迟时间，用户 v 发出行为 a 后，用户 u 在 $t_{v,u}$ 的延迟时间后被影响而发出同样的 a 行为。$t_{v,u}$ 是通过历史微博记录统计出的行为从用户 v 传播到用户 u 的平均延迟时间。用户 v 和用户 u 之间行为传播平均延迟时间定义如公式（10-13）所示：

$$t_{v,u} = \frac{\sum_{a \in A}(t_u(a) - t_v(a))}{A_{v2u}} \qquad \text{式（10-13）}$$

公式 10-13 中，$t_u(a)$ 表示用户 u 发出行为 a 的时间，A 表示微博中的历史行为集合。

前文中已假设任意用户的所有父节点对该用户的影响都是独立的，父节点对该用户的影响没有依赖关系，因此，如果能计算任一父节点对该用户的影响概率，就可以通过式（10-12）计算该用户受到所有父节点的行为影响联合概率，即该用户发出同样行为的概率。本节提出的 Jaccard 系数连续时间模型不同于其他文献中的静态模型，静态模型里假设用户之间的影响概率是静态的，不随时间推移而变化，而根据真实社交网络的动态性，用户之间的影响概率应该是一个连续时间函数。下面先介绍基于微博的行为影响概率 Jaccard 系数模型，再引入用户间影响概率的时间特性，提出 Jaccard 系数连续时间模型。

1. Jaccard 系数模型

Jaccard 系数，又叫做 Jaccard 相似性系数，用来比较样本集中的相似性和分散性的一个概率，它只关心个体间共同具有的特征是否一致。Jaccard 系数等于样本集交集与样本集并集的比值，简单而言，集合 A 和集合 B 的 Jaccard 系数等于集合 A 和集合 B 中共同拥有的元素数与集合 A、B 总共拥有的元素数的比例，公式如式（10-14）所示：

$$\text{Jaccard}(A,B) = \frac{A \cap B}{A \cup B} \qquad \text{式（10-14）}$$

很显然，Jaccard 系数值区间为[0,1]。由于 Jaccard 系数经常用于衡量两个样本集的相似度，因此就采用 Jaccard 系数来评估两个用户之间行为的相似度，相似度高表示用户 v 对用户 u 的行为影响概率比较大。针对 Jaccard 系数一般公式，这里提出的用户间的行为影响概率公式如式 10-15 所示。

$$p_{v,u} = \frac{A_{v2u}}{A_{v|u}} \qquad \text{式（10-15）}$$

然而，值得注意的是，微博中的某个用户 u 被影响而发出某个行为，这个行为可能是被之前已发出过相同行为的所有父节点共同影响而激发的。因此，用户 u 的这些父节点中的每个节点都

只贡献了联合影响中的一部分。假设用户 u 在时间$t_u(a)$时发出行为 a，S 是影响用户 u 发出行为 a 的所有父节点集合，从而有$\forall v \in S, \mathcal{T}(v,u) \leq t_v(a) \leq t_u(a)$。在部分计分 Jaccard 系数模型中，集合 S 中的每个父节点根据权威值的比重计算使用户 u 发出行为 a 的功劳，这也是计算主题用户权威值的意义。一般的，对于行为 a，用户 u 的每个已发出行为 a 的父节点的部分计分定义为：

$$credit_{v,u}(a) = \frac{W(v)}{\sum_{z \in X} W(z)} \qquad \text{式（10-16）}$$

式（10-16）中，W（v）表示用户 v 的权威值，计算过程已在 10.4.2 节中详细介绍。集合 X 是用户 u 的父节点中，所有已发出行为 a 的节点集合，$X = \{z | t_z(a) < t_u(a)\}$。

因此，在部分计分 Jaccard 系数模型中，用户 v 对用户 u 的行为影响概率可以使用公式（10-17）计算：

$$p_{v,u} = \frac{\sum_a credit_{v,u}(a)}{A_{v|u}} \qquad \text{式（10-17）}$$

式（10-17）中，A 是微博社交网络的历史微博集合。

2. Jaccard 系数连续时间模型

真实社交网络中，两用户之间的影响概率并不是与时间无关的常数。很明显，当用户 u 发现他的某个父节点刚刚发表了某个行为 a，用户 u 必定会有很强烈的兴趣和意愿去了解这个行为，并可能会发出同样的行为。但是随着用户 u 对父节点发出行为 a 的发现越来越晚，用户 u 的这种意愿就会大幅度衰减。这里对 Twitter 历史微博进行统计，相同行为数随时间间隔变化如图 10.22 所示。

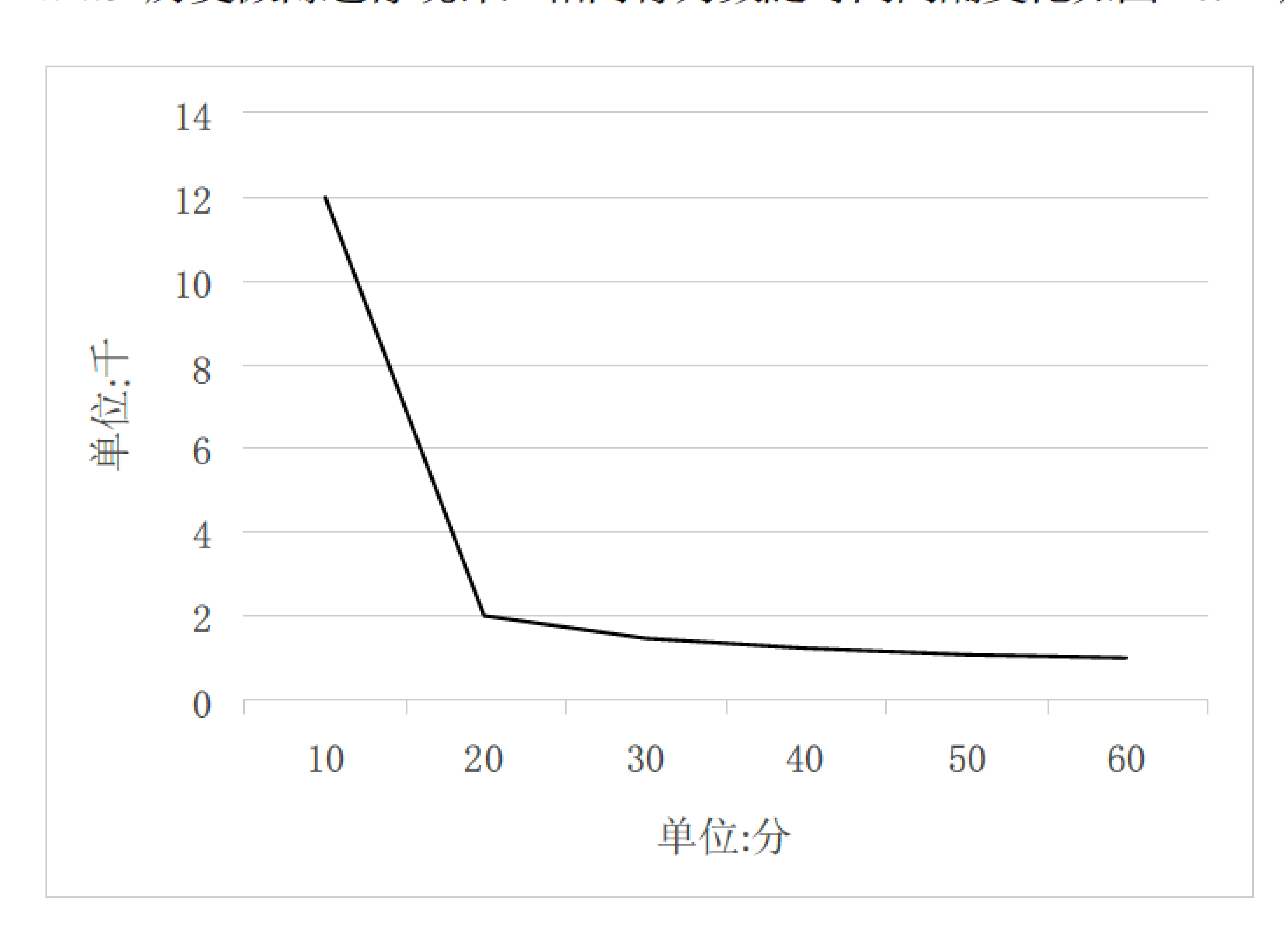


图 10.22　Twitter 中不同时间间隔发出的相同行为数

由图 10.22 可以看出，用户受影响的概率是随时间变化指数衰减的，因此，用户 v 对于用户 u 的影响概率其实是关于时间的函数，在 Jaccard 系数连续时间模型里，用户 v 对于用户 u 的行为影响概率应该为$p^t_{v,u}$，其定义公式如式（10-18）。

$$p_{v,u}^{t} = p_{v,u}^{0} e^{-(t-t_v)/t_{v,u}}$$ 式（10-18）

$p_{v,u}^{0}$是用户 v 对用户 u 的影响概率的最大值，在这个指数衰减的模型里，将用户 v 发表行为 a 的时刻t_v代入上式，用户 v 对用户 u 的影响概率达到最大值$p_{v,u}^{0}$，最大影响概率$p_{v,u}^{0}$即上文提到的$p_{v,u}$。公式中的$t_{v,u}$是用户 v 对于用户 u 影响的平均延迟时间。

由上可得，在 Jaccard 系数连续时间模型中，任意两个用户在任意时刻的影响概率$p_{v,u}^{t}$可以计算得到，因此，用户 u 所受到所有父节点的影响联合概率就可以通过公式（10-19）计算。

$$p_u^t(S) = 1 - \prod_{v \in S}\left(1 - p_{v,u}^{t}\right)$$ 式（10-19）

在这个连续时间模型中，用户 u 受到的影响联合概率是与时间有关的函数，它随时间的推移而变化。当用户 u 某个父节点在某一时刻被激活从而对用户 u 具有影响力时，用户 u 受到的影响联合概率将会急剧变大，函数曲线将会迅速上升。因此，函数$p_u^t(S)$是一个分段连续函数，这也说明该函数的计算和更新是比较复杂的，需要有一个类似的更为简单并能保持精确性的改进模型。

本系统提出的用户影响联合概率计算所需的训练集是 Twitter 中用户发出的所有股票微博信息，并不考虑股票无关言论。训练集的选定使用户所受到的影响联合概率的计算更加精确，能够更准确地预测股票价格涨跌趋势。

计算所需要的训练集全部通过 Twitter API 采集，由于 Twitter 具有海量的用户，每天都会产生数以亿计的微博，所以这里提出的计算方法必须能够面对海量数据处理带来的挑战，尽可能地提高时间效率，因此它的计算过程必须尽可能减少扫描训练集次数。

用户影响联合概率计算的输入由社交网络有向图 G=（V,E,T）以及用户行为历史记录组成，而用户行为历史中的每一条记录则是一个三元组（u,a,t_u），表示用户 u 在 t_u时刻发出了行为 a。根据上文的实现目标，本系统要求输入训练集必须满足按行为 id 排序，具有形同 id 的行为则按时间先后排序。由于输入训练集以数据库方式保存，在判断行为相似度之后，数据库提供的排序功能很容易满足排序需求。

针对上文提出的 Jaccard 系数连续时间模型，只需要对训练集作一次遍历即可。算法中使用表 current_table 保存关于行为 a 的所有三元组（u,a, t_u），并且三元组在表中是以发出时间 t 排序的。当从训练集中读出一个元组（u,a,t_u）时，表明用户 u 在 t_u时刻发出了行为 a，这时可以从行为 a 的 current_table 中查询 u 的所有父节点，这些父节点发出行为 a 的时间先于用户 u，换句话说，正因为这些父节点发出行为 a，才影响用户 u 成为行为 a 的传播节点。E^{t_v} 表示在 t_v时刻微博有向图网络中所有用户关系形成的边集合，E^{t_v} 的引入是为了保证当判断用户 v 将行为传播到用户 u 时， 节点 v 和节点 u 一定是连通的。E^{t_v} 集合随着微博用户关系链的连接和断开而动态更新。

下面先介绍 Jaccard 系数连续时间模型的实现算法。用户影响联合概率的计算重点在于先计算微博中两用户间的行为影响概率，得到用户间行为影响概率后就可以通过式 10-19 直接计算用户的影响联合概率。具体的计算用户间行为影响概率的算法伪代码如下所示：

```
for each aciton a do
  current_table = Ø
```

```
3:  for each user tuple < u, a, t_u > do
4:      increment A_u
5:      parents = Ø
6:      for each user v: (v, a, t_v) ∈ current_table && (v, u) ∈ E^{t_v} do
7:          if t_u > t_v then
8:              increment A_{v2u}
9:              update T_{v,u}
10:             insert v in parents
11:         increment A_{v&u}
12:     for each parent v ∈ parents do
13:         update credit_{v,u}
14:     add (u, a, t_u) in current_talbe
```

算法 1~6 行比较简单，算法第 7 行忽略两个用户在同一时刻发表了行为 a 的情况，当这种情况发生时，信息在两用户之间是否发生了传播是不确定的，因此第 7 行使用$t_u > t_v$作限制。当分析出所有发出行为 a 的节点时（表 current_table 中的节点），算法 8~11 行对用户 u 相关的所有参数均进行更新。

算法中最重要的是$credit_{v,u}$的更新，它是计算用户之间行为影响概率的关键。首先要得到影响用户 u 成为传播节点的所有父节点，算法第 10 行将这些节点插入到 parents 集合中。算法第 12 行对 parents 集合进行遍历，通过上文中的评分公式（10-16）对$credit_{v,u}$进行更新（算法第 13 行）。最后，算法第 14 行将关于行为 a 的元组（u,a,t_u）添加到关于行为 a 的 current_table 表中。

得到$credit_{v,u}$值后，可以根据公式（10-17）计算用户 v 和用户 u 之间的行为影响概率，进而可以计算出用户 u 受到的影响联合概率。通过前面的算法介绍可以看出，对于 Jaccard 系数连续时间模型，用户的影响联合概率的计算只需要对训练集进行一次遍历。

3. Jaccard 系数离散时间模型

Twitter 作为全球最大的在线社交网络，每天贡献的数据量是十分庞大的。北京时间 2012 年 12 月 19 日凌晨，Twitter 宣布其月活跃用户已破 2 亿，较同年 3 月份的 1.4 亿活跃用户相比增加了 6000 万，9 个月增长了 42%，每天发布信息早已达到 3.4 亿条。改进模型需要面对如此庞大的海量数据处理，因此模型必须要能够对每个用户受到的影响联合概率进行快速的计算。于是，当用户 u 的某个父节点 w 被激活而对用户 u 产生新的影响时，用户 u 受到的影响联合概率需要被快速计算。为了做到快速的计算，在用户 w 被激活后，用户 u 的影响联合概率$p_u(S\cup\{w\})$需要能够被增量式更新，而不需要再次使用公式（10-12）重新访问所有已激活的父节点对用户 u 的影响概率。

上面提出的 Jaccard 系数连续时间模型中，由公式（10-19）可以明显看出，该模型不满足增量式更新特性。一旦用户 u 有一个父节点被激活，为计算用户 u 受到的影响联合概率，需要重新计算$p_u^t(S)$。如果用户 u 的父节点集合大小为 d，则用户 u 的影响联合概率函数会有局部极大值，在处理海量微博数据时将会产生非常巨大的时间开销。

因此，在本小节提出一个与 Jaccard 系数连续时间模型接近的新的模型——Jaccard 系数离散

时间模型，它满足增量式更新，而且也符合微博中两用户之间的影响概率随时间推移而衰减的特性。在离散时间模型中，假设用户 v 对用户 u 的影响概率$p^t_{v,u}$在时间间隔为$t_{v,u}$内保持为常数$p_{v,u}$，当超出这个时间间隔后，影响概率$p^t_{v,u}$直接降为 0。也就是说，用户 v 对于用户 u 的影响概率在$[t_v, t_v+t_{v,u}]$区间内保持为常数$p^0_{v,u}$。离散时间模型中的这个设定将会使该模型满足增量式更新特性，从而可以更好地应用于海量数据处理之上。

当用户 u 的某个父节点 w 突然被激活而对用户 u 具有行为影响时，用户 u 受到的影响联合概率$p_u(S \cup \{w\})$可以通过公式（10-20）进行增量式更新。

$$
\begin{aligned}
p_u(SU\{w\}) &= 1-(1-p_{w,u})*\prod_{v\in s}(1-p_{v,u}) \\
&= 1-(1-p_{w,u})(1-p_u(s)) \\
&= p_u(s)+(1-p_u(s))*p_{w,u}
\end{aligned}
\qquad \text{式（10-20）}
$$

但是由于 Jaccard 系数离散时间模型的特性，用户 u 的某个父节点 w 在滑过长为$t_{w,u}$的时间窗后，他对用户 u 的影响概率降为 0，这时用户 u 受到的影响联合概率$p_u(s/\{w\})$也可以进行增量式的更新，如公式（10-21）所示：

$$
\begin{aligned}
p_u\left(\frac{s}{\{w\}}\right) &= 1-\frac{\prod_{v\in s}(1-p_{v,u})}{(1-p_{w,u})} \\
&= 1-\frac{1-p_u(s)}{1-p_{w,u}} \\
&= \frac{p_u(s)-p_{w,u}}{1-p_{w,u}}
\end{aligned}
\qquad \text{式（10-21）}
$$

Jaccard 系数离散时间模型相对于连续时间模型，在计算任意两用户之间的影响概率时，用户 v 对用户 u 的计算公式（10-16）中的 X 集合将更改为$X=\{z|0<t_u(a)-t_z(a)<t_{v,u}\}$。Jaccard 系数离散时间模型得到的影响联合概率与上述的 Jaccard 系数连续时间模型十分接近，并且与连续时间模型相比，离散时间模型中的影响联合概率可以被增量式更新，因此在对微博等在线社交网络中的海量数据处理时，更具备高效性。

Jaccard 系数离散时间模型假设了用户之间的行为影响概率是非静态的，它在影响平均延迟时间$t_{v,u}$内保持常量$p^0_{v,u}$，而超出这个时间间隔后，用户 v 对用户 u 的影响概率迅速降为 0。所以用户的行为影响概率计算过程中，还需要对$t_{v,u}$加以考虑。从前面算法可以看出，$t_{v,u}$的计算需要对训练集先作一次遍历，所以对于 Jaccard 系数离散时间模型，影响联合概率的计算总共需要对训练集做两次遍历。具体算法的伪代码如下所示：

```
for each aciton a do
    current_table = Ø
    for each user tuple < u, a, t_u > do
        parents = Ø
```

```
5:      for each user v: (v,a,t_v) ∈ current_table && (v,u) ∈ E^{t_v} do
6:          if 0 < t_u − t_v < t_{v,u} then
7:              increment A_{v2u}
8:              insert v in parents
9:      for each parent v ∈ parents do
10:         update credit_{v,u}^{t_{v,u}}
11:     add (u,a,t_u) in current_talbe
```

离散时间模型的算法伪代码与连续时间模型的算法基本相同，仅仅在第 6 行中有较大差别。算法第 6 行中需要保证用户 v 和用户 u 发出相同行为 a 的时间间隔不能超过$t_{v,u}$，这个限制条件是 Jaccard 系数离散时间模型的关键。此算法得到的$credit_{v,u}^{t_{v,u}}$值，根据公式（10-17）计算用户 v 和用户 u 之间的行为影响概率，然后根据公式（10-19）计算用户 u 受到的影响联合概率。

10.4.4　预测股票价格涨跌模块详细设计

该模块主要分为两部分：微博情感信息传播预测模型的设计和股票价格涨跌趋势预测的设计。

1. 微博情感信息传播预测模型

社交网络的信息传播模型一定具有两个要点：传播规则和网络拓扑结构。传播规则指社交网络中的用户如何接受消息并把消息传播到其他用户，而社交网络中的信息传播一定有其传播路径，微博在线社交网络的信息传播路径就是微博中好友关系形成的有向图。

微博网络中，某用户发表一条有关股票的微博后，该用户的所有粉丝会通过各种终端设备获得这一消息。感兴趣的粉丝会尽快对这一消息进行评论或转发，从而将该消息分享给更多的用户。微博中的信息在传播时，不存在明显的限制或围栏，信息传播的深度和广度并不仅仅取决于微博用户的权威值、信息的影响力，还取决于粉丝、粉丝的粉丝等人对信息传播的贡献。粉丝的数量和质量，他们对信息的传播能力和意愿也是决定信息传播程度的重要因素。

由于用户发出的有关股票的微博是有一定情感的，例如看涨或者看跌某支股票，那么这种情感会在用户网络中不断传播下去，在微博的情感传播路径中，接收到信息的用户受到了信息的影响，成为传播用户，然后将接收的信息情感传播到所有的子节点。而接收到信息的节点没有成为传播节点的原因是父节点的信息对该节点的影响并不足以改变他的意愿和行为。传播节点可能是传播路径的中间节点，也可能是传播路径的终止节点。传播节点成为传播路径的终止节点是因为该节点对所有子节点的影响都没有成功，也就是所有子节点虽然接收到了来自于他的传播信息，但是均没有改变自己的状态，因此终止节点不会影响下一时刻的传播。

社交网络中信息传播的网络拓扑结构用有向图$G = (V,E,\mathcal{T})$表示，V 表示社交网络中的用户集合，E 表示社交网络中用户之间的关注关系，$\mathcal{T}$标明了有向图中的每条边的建立时间。社交网络中的信息就在有向图 G 的边集 E 上传播。

图 10.23 为 Twitter 社交网络中信息传播预测模型结构图，在初始时刻 t，所有已发出行为 a

的用户用黑色节点表示，而没有发出行为 a 的用户用白色节点表示，节点间的连线表示用户间的关注关系。当关系网络图中有黑色节点出现时，黑色节点就对邻接的白色子节点产生影响。白色子节点根据自身所受到的影响联合概率决定发出或者不发出行为 a，而且发出行为 a 的时间也是可以预测的，白色节点成为传播节点的时间是一个区间[b，e]。

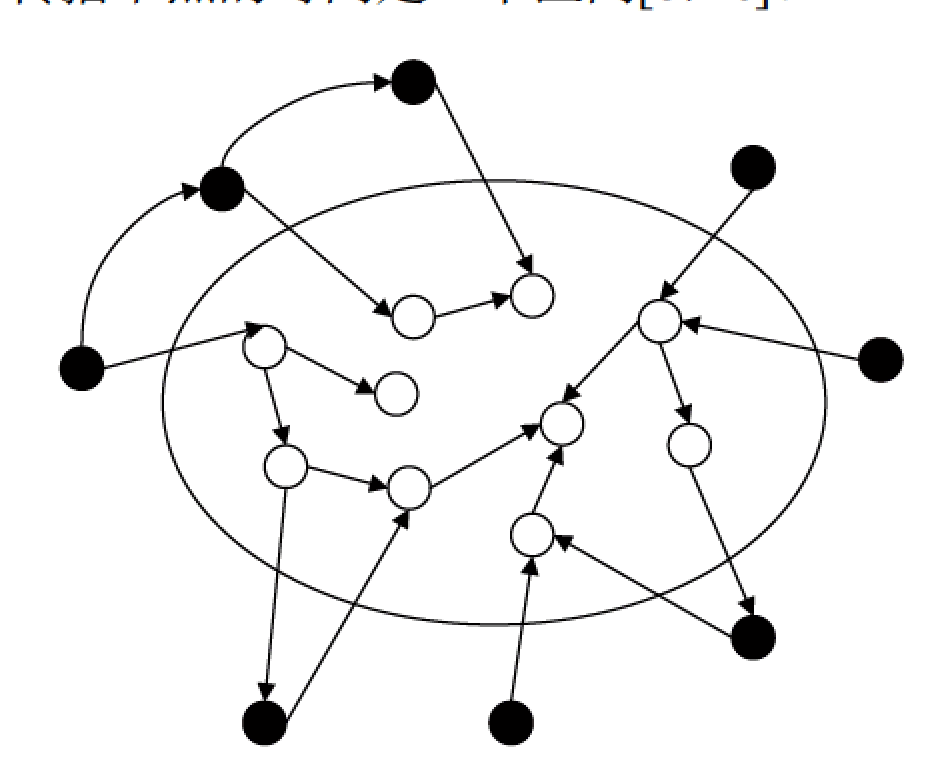


图 10.23　信息传播预测结构图

当微博中用户 u 的某父节点 v 在 t 时刻被激活发出行为 a 后，用户 u 受到行为 a 的影响联合概率函数曲线会快速上升，如图 10.22 所示。从图 10.22 中可以直观看出，用户 u 受到的影响联合概率函数是一个分段非连续函数。当影响联合概率 $p_u(s) > \theta_u$ 时，用户 u 就有可能被激活而发出相同行为 a。这里就选取联合影响概率函数值大于 θ_u 的第一个局部极大值对应的时间 t，作为预测用户 u 成为传播节点的时间区间的左值。当到达左值时刻时，用户 u 就进入易感染状态，并随时可能被激活，成为传播节点。

本系统采用微博中用户 v 对用户 u 影响的半衰期作为预测用户 u 成为传播节点的时间区间的右值。由 10.4.3 节中的介绍可知，用户 v 对于用户 u 影响的平均延迟时间为 $t_{v,u}$，在用户 v 对用户 u 影响的半衰期内，更精确来说就是 $t_{v,u} * \ln(2)$ 时间内，用户 v 发出行为平均有 50%的概率会传播到用户 u，因此，本文就采用影响延迟的半衰期 $t_{v,u} * \ln(2)$ 作为预测用户 u 成为传播节点的时间区间的右值。

依前所述，用户 u 受到影响成为传播节点的时间是一个区间[b,e]，区间的左值是用户 u 的影响联合概率达到 θ_u 的时刻，区间的右值是最近被激活的父节点对自身影响延迟的半衰期。但是，用户 u 成为传播节点的时间区间在信息传播模型中处理是比较复杂的，因此对用户 u 成为传播节点的时间需要做精确的预测。在社交网络的病毒营销应用中，因为用户 u 总是会提前发出行为 a，所以用户 u 成为传播节点的时间左值 t 是不确定的。用户 u 虽然提前发出行为 a，但是对于信息传播模型没有影响，所以这里不采用时间区间的左值，而采用时间区间的右值作为预测用户 u 成为传播节点的精确时间。

在情感信息传播预测模型中，微博信息基于 BFS 的传播预测步骤为：

步骤 01　从图 10.23 中的黑色节点开始，遍历所有黑色节点，逐一加入到队列中；

步骤 02　取出队列头节点（黑色节点），访问黑色节点的各个邻接白色子节点，对邻接白色子节点做出行为 a 的发出预测，如 10.4.3 节中介绍的。预测会发出行为 a 的白色节点转为黑

色节点，预测发出时间小于指定时间，则加入到队列尾中。

步骤 03　重复步骤（2），直到黑色节点队列为空。

在初始时刻 t，所有已经发出行为 a 的用户用集合 B^t 表示。未发出行为 a 的用户会根据自身受到的影响联合概率决定自己的行为发出与否。当用户受到的影响联合概率大于该用户成为传播节点的阈值时，用户就会发出行为 a，该用户被添加到集合 B^{t+1} 中。反之，用户的行为状态不会发生变化，行为 a 的用户集合 B^{t+1} 也就不会发生变化。

对于行为 a，父节点对用户 u 的影响联合概率用 $p_u(s)$ 表示，则信息传播模型是一个四元组 $(G, p_u(s), \theta_u, B^t)$,。其中 G 就是信息在微博中传播的有向图 $G=(V,E,J)$， B^t 的定义如下：

$B^0=\{u,v\cdots\}$，初始时刻节点集合 V 中发出行为 a 的节点；

$B^t=B^{t-1}US^t, t\geqslant 1$

其中，$S^t=\{u|u\in V$,（v,u） $\in E, v\in B^t, u\notin B^{t-1}, p_u(S)>\theta_u\}$。

情感信息传播预测的过程类似于有向图的广度优先遍历，在遍历过程中会增加时间的限制。当预测某一时刻的网络状态时，根据节点的发出预测时间而终止基于 BFS 的信息传播预测。当到达终止时刻前，行为 a 在网络中传播可能也会有终止状态，未达到终止状态时，网络中发出行为 a 的用户会逐渐增加，体现在集合B^t上就是集合的大小随时间推移而增大，当达到终止稳定状态时，网络中所有用户对于行为 a 的发出状态不再发生变化，集合B^t的大小也不会发生变化。

2. 股票价格涨跌趋势预测模型

上面提出的基于微博情感信息传播预测模型，对于股票价格的预测十分适用。在股票价格预测的实现中，提取股市开盘时某支股票相关的微博言论，通过信息传播预测模型，预测在股市收盘前对这支股票持看涨态度和看跌态度的用户数量以及在网络中的分布情况，可以很清晰地对未来股票价格涨跌趋势做出预测。纽约证券交易所是美国第一大证券交易市场，它的交易时间是从美国东部时间上午 9:30 到下午 4:00，中午不休息，美国其他证券交易所，如纳斯达克证券交易所均遵循上述时间。这里只针对美国股市对 Twitter 平台的微博内容进行分析，基于新浪微博等中文社交网络的股票价格预测方法与 Twitter 平台类似。

在预测股票价格涨跌趋势过程中，采集美国股市上午开市时的股票言论微博，分析微博中体现的对该股票的情感态度，通过上节中介绍的用户影响联合概率的计算方法，逐步预测网络中的用户在两种情感态度影响下的行为，并在美国股市下午收市前，比较网络中持不同情感态度的用户数量，再根据“个人行为受情感支配”理论，可以预测这支股票在股市收市前是涨还是跌。

预测股票价格看涨传播趋势的流程图如图 10.24 所示。

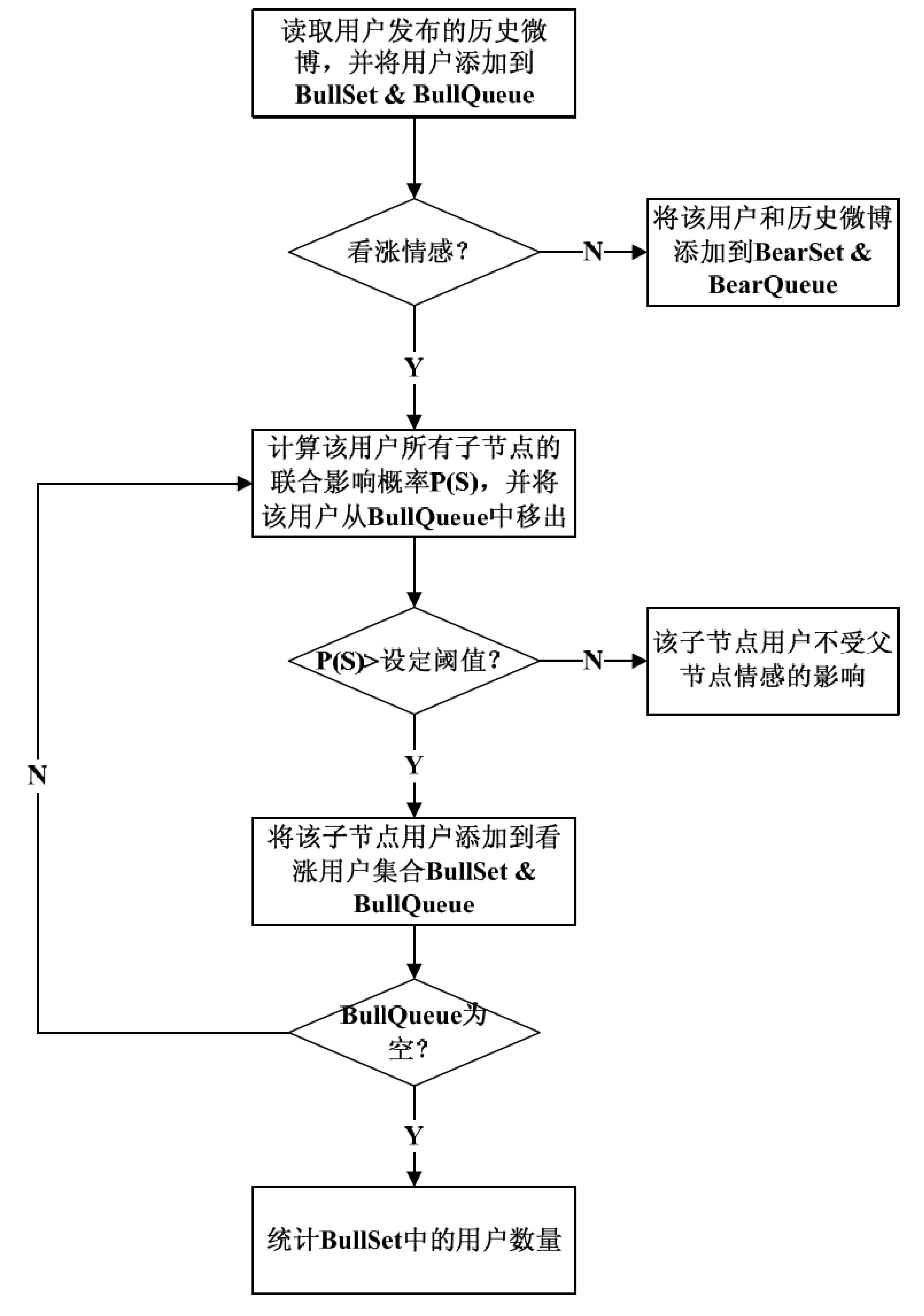


图 10.24　预测看涨情感传播趋势流程图

具体算法的详细描述如下，预测股票价格涨跌趋势算法以苹果公司股票$AAPL 为例，预测对$AAPL 看涨的情感在网络中的传播趋势，而看跌情感的传播完全相同。算法的输入是社交网络有向图$G=(V,E,T)$以及股票开市时间T_1时 Twitter 平台上对$AAPL 持看涨的情感态度的用户节点。所有对$AAPL 持看涨态度的用户用集合 BullSet 表示，而每次预测的看涨用户用队列 BullQueue 表示，预测用户 u 发出看涨态度行为的时间为t_u，当且仅当$t_u < T_2$（收盘时间）时，用户 u 被加入看涨用户集合 BullSet 以及队列 BullQueue 中。整个预测算法过程类似于有向图的深度优先遍历，其伪代码描述如下所示：

```
1:  BullSet = Ø, BullQueue = Ø
2:  for each bullish node v at time T_1 do
3:      add v to BullSet & BullQueue
4:  while BullQueue is not empty do
5:      delete the front node v
6:      for each child_node u of v do
7:          if u is not bullish node and t_v + t_{v,u} < T_2 then
8:              calculate p_u(S)
9:          if p_u(S) > θ_u then
10:             t_u = t_v + t_{v,u}
11:             add u to BullSet & BullQueue
12: count |BullSet|
```

算法具体的步骤如下：

步骤 01　初始化看涨用户集合 BullSet 和新增加看涨用户集合 BullQueue（第 1 行）。

步骤 02　将开盘时刻网络中所有看涨用户添加到两个集合中（第 2~3 行）。

步骤 03　对新增加看涨用户集合中的每个用户进行子节点分析。

- 看涨用户的子节点如果不是看涨用户且成为传播节点的时间先于收盘时间T_2，计算该节点的影响联合概率（第 7~8 行）。
- 节点的影响联合概率大于节点的行为影响阈值θ_u，更新该节点的行为发出时间，添加该节点到 BullSet 和 BullQueue 两个集合（第 9~11 行）。

步骤 04　统计看涨用户集合中用户数量（第 12 行）。

以上算法为预测看涨用户在股市收盘时的数量，利用同样的方法可以预测看跌用户在股市收盘时的数目，只需将算法第 2 行股市开盘时对股票持看涨态度的用户节点变为持有看跌情感的用户节点即可。预测看跌情感传播趋势流程图如图 10.25 所示。

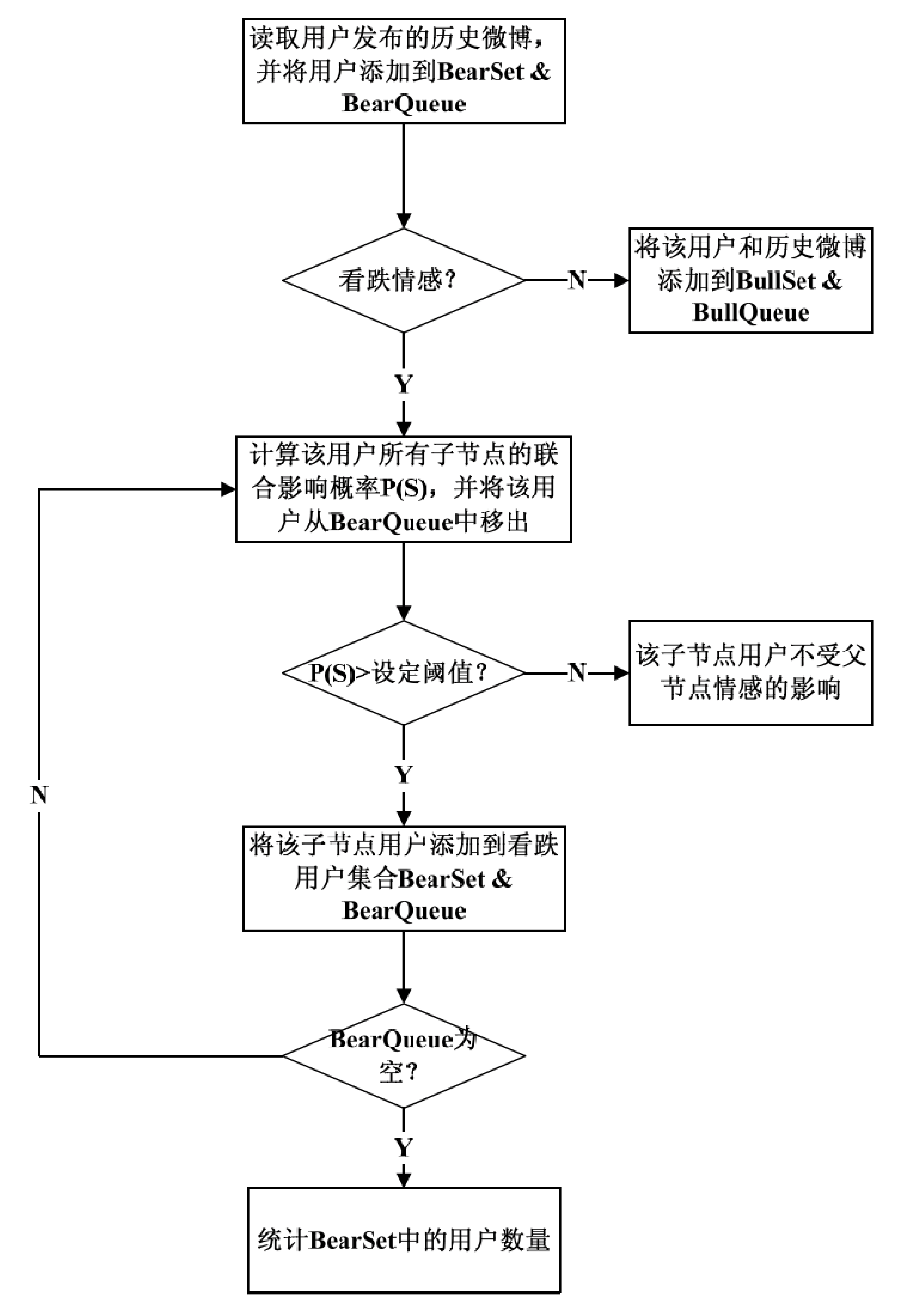


图 10.25　预测看跌情感传播趋势流程图

比较两种情感态度持有的用户数量，可以直观地预测出微博网络中大多数用户对股票价格的情感态度，由于大多数用户缺乏独立思考，具有从众的心理，且个人的行为受情感的支配，大多数人的情感指引他们对股票采用相同的操作，个人的行为演变为网络整体的行为，从而使得股票的价格随着大多数用户的心理趋势而变化。因此这里从预测 Twitter 社交网络中大多数人对股票的情感态度，成功预测这支股票在收盘时刻的价格涨跌趋势。预测的流程图如图 10.26 所示。

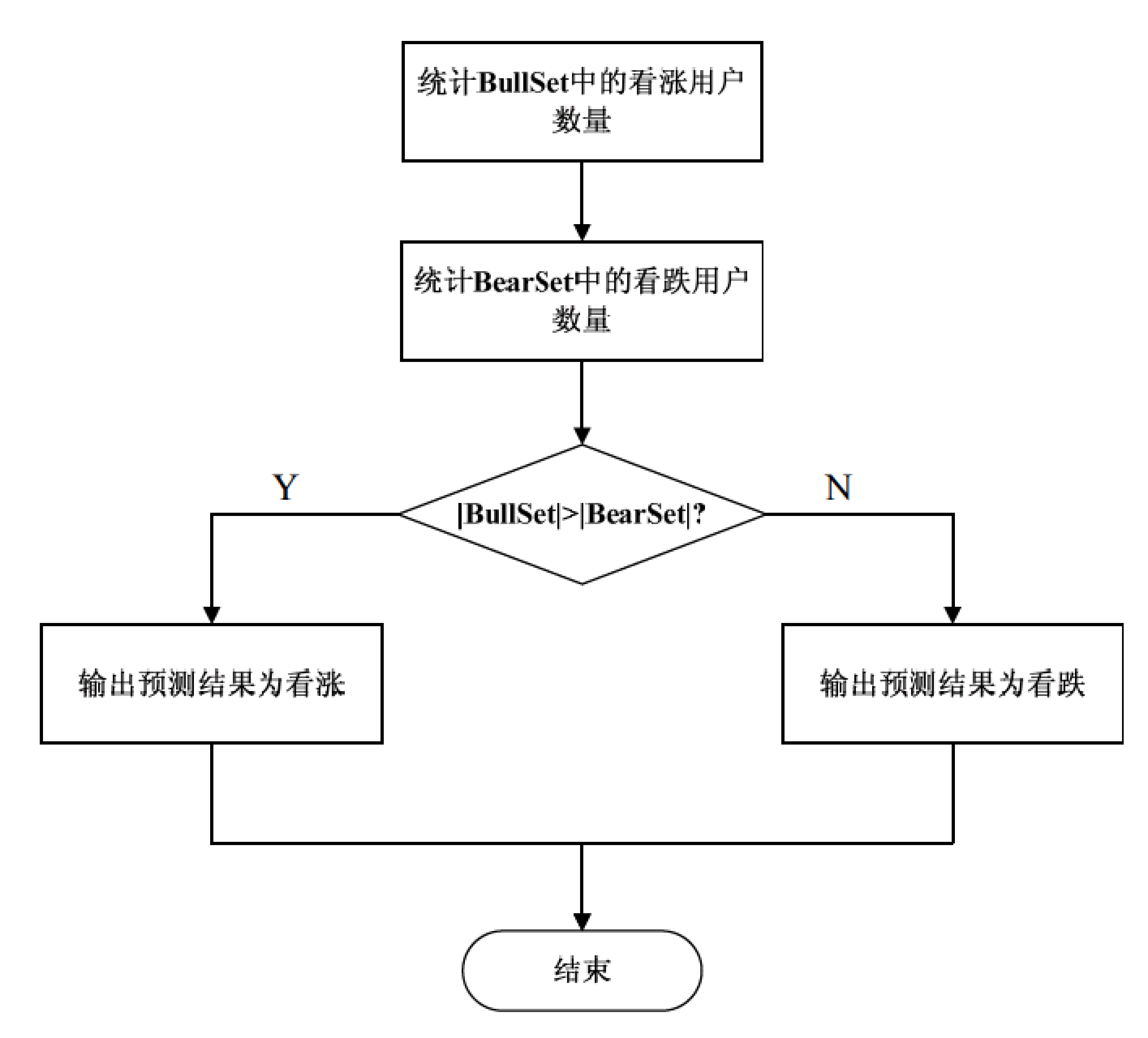


图 10.26 股票价格涨跌预测流程图

由股票价格预测的方法可以看出，Twitter 网络中对股票持看涨或看跌情感的源头是该股票开盘时微博网络中首先对股票持看涨或看跌情感的用户。当源头用户越多，情感越强烈时，对股票价格预测的准确率也会越高。另外存在一种情况，当股市收盘前某一时刻，社交网络中所有用户的情感可能就不再发生更新，这时持两种情感的用户的数量对比就是最终股市收盘时的用户数量对比，可以直接通过此时用户的数量预测收盘时股票价格的涨跌趋势。

10.4.5 系统实现

本系统主要提供了一个基于微博的股票市场预测软件，通过该系统用户可以采集并查询来自 Twitter 有关股票的微博信息，系统通过对信息进行一定的处理和分析，根据微博情感在社交网络中的传播趋势来预测未来一段时间内股票价格的涨跌趋势。

股票市场中的交易是一个不稳定的时刻动态变化的过程，不但受到国内外的经济因素影响，还受到股票投资人的行为控制。除此之外，来自政府的宏观调控也是影响股票价格在未来走势的主要因素。基于以上原因，股票价格涨跌趋势预测选取的预测对象是价格运行较平稳的股票，这种模型可以反映股市的正常交易规律，从而具有良好的推广能力。综上所述，本系统选取苹果公司股票$AAPL 以及 Google（谷歌）公司股票 GOOG 作为预测对象，选取的两个预测股票对象流通市值较大，企业业绩较好，而且企业的诚信度较高，该选取尽量避免了人为操纵股票和弄虚作假的情况。

股票市场的交易运作是周期性的，可以大致分为长期、中期、中短期、短期等几个周期。若进行不同周期长度的股市预测，则需要应用不同规模的数据。预测周期长则需要大规模的数据，而预测周期短，数据需求规模就小一些。数据规模的选取与预测对象的选取同等重要，当应用的

数据规模较大时，预测模型具有较强的推广能力，只是在局部趋势预测的精度会差一些；但是应用数据规模较小时，预测模型不具备较强的推广能力。综上所述，本系统针对中短期预测，预测苹果公司股票$AAPL，选取 Twitter 平台上对$AAPL 感兴趣的用户网络，该网络中有 5587 名用户，历史发布关于$AAPL 的微博言论共 82 760 条；预测 Google 公司股票$GOOG 则选取对$GOOG 感兴趣的用户网络，网络中有 3250 名用户，历史发布关于$GOOG 的微博言论共 36 802 条。

对股票价格涨跌趋势的预测以苹果公司股票$AAPL 为例，首先，通过 Twitter 平台提供的 API，提取 2011 年 8 月 1 日至 2011 年 12 月 21 日时间段内，Twitter 用户所有发表内容包含$AAPL 的微博。搜索包含关键词“$AAPL”的微博采用 GET search/tweets API，访问 https://api.twitter.com/1.1/search/tweets.xml?q=$AAPL，将返回的 XML 格式的数据解析即可得到近期微博内的包含“$AAPL”关键字的微博信息。

提取的$AAPL 股票内容存储在 MySQL 数据库中的表$aapl 中（股票的名称即为数据库的表名，有助于区分不同股票的微博内容表），表$aapl 的结构与返回 XML 格式信息中的属性基本相同，如图 10.27 所示。

id	text	source	user_id	user_name	creat_at
100000131646107648	"@KMVarrichio: Actually, that tweet	<a href="http://twitter.co	377458120	highwhilesober	2011-08-07 00:27:43
100002052985454592	Tech titans hoard cash and don't re	<a href="http://twitter.co	16413382	CapitalObserver	2011-08-07 00:35:22
100003517393145857	RT @SAI: Here Are The Best iPhone	<a href="http://www.bus	309349384	qneteg	2011-08-07 00:41:11
100005962567532544	RT @CapitalObserver: Tech titans h	<a href="http://twitter.co	19302828	deandobbs	2011-08-07 00:50:54
100008591855058944	8/5/11 Algorithm tracks 611 stocks	<a href="http://twitter.co	212467856	strategic_opt	2011-08-07 01:01:21
100008966645493760	It figures that on the day I downloa	<a href="http://itunes.ap	171878775	JamarParris	2011-08-07 01:02:50
100011374054350849	$AAPL Thanks. Somehow I knew y	<a href="http://twitterfee	341344687	stockpulses	2011-08-07 01:12:24
100013465581793280	GDP = Gross Domestic Propaganda	<a href="http://mobile.tv	305111045	party_man2012	2011-08-07 01:20:43
100015269082169344	RT @upsidetrader: NEW POST: COI	<a href="http://ubersoci	17533601	skayfe	2011-08-07 01:27:53
100016754310389760	Is it 1984? Because $aapl wants to t	<a href="http://mobile.tv	170432370	optionspirate	2011-08-07 01:33:47
100018285105520643	@elingford on $IDCC .. Tech relatec	<a href="http://twitter.co	66232446	BreakoutBull	2011-08-07 01:39:52
100019252114882561	$AAPL You are so smart - why can't	<a href="http://twitterfee	341344687	stockpulses	2011-08-07 01:43:42
100020055164727296	? Your Daily $AAPL ? is out! http://b	<a href="http://paper.li"	17433343	DigDugTrader	2011-08-07 01:46:54
100020478533578752	RT @DigDugTrader: ? Your Daily $A	<a href="http://paper.li"	37928746	Hephaestus7	2011-08-07 01:48:35

图 10.27　$AAPL 股票相关微博数据库表$aapl

苹果股票相关微博表$aapl 中，每个元组的属性依次为：Twitter 微博 ID、微博文本内容、微博来源、发出微博的用户 ID、用户名、发出微博时间。本文使用 XML 解析器来解析 API 请求返回的关键字搜索结果，从结果中解析以上各属性对应的值，直接插入数据库的$aapl 表中。关键字搜索结果以时间顺序排列，因此也保证了$aapl 表中所有的微博内容同样按照时间顺序排列。之后的 API 搜索请求，可以在访问资源地址中加入 since_id 属性以保证搜索结果不会与之前的搜索结果重复，如：https://api.twitter.com/1.1/search/tweets.xml?q=$AAPL&since_id=100020478533578753。

当提取完$AAPL 股票相关言论的微博内容后，根据所提取的微博内容，建立对苹果公司股票感兴趣的用户网络。首先根据$aapl 表中的 user_id 列，通过 GET users/show API 获取用户的详细信息。获取 ID 为 377458120 的用户信息的访问资源地址为 http://api.twitter.com/1.1/users/show.xml?user_id=377458120，再次通过解析 XML 格式的返回数据，将用户信息资料存储到数据库表 user_$aapl 中，如图 10.28 所示。

id	name	screenName	location	description	langu	follower	user_follower	followin	user_following	weight
100070861	Scot Kramarich	scotkramarich	Los Angeles	Software developer	en	142	137508847+1778	242	224180785+3152	20.33
100098265	Ryan Rust	Ryan_Rust	Hampton Road		en	38	94267686+49202	150	393929229+4920	7.65
10011032	skycrusher	skycrusher	On a coast som	Life & Golf Are Not A	en	6	238526507+2281	51	369732560+1165	3.56
100244944	Gerald Thurman	compufoo	Tempe, Arizona	Computing/math inst	en	59	83980008+17784	0		9.49
100286145	Gino Giacumbo	GinoInvesting	San Diego, CA	Obscure fascination v	en	99	391175224+3047	140	15298655+87738	11.45
100316615	Andrew Rhombe	arhomberg	London	Founder of Jellybook	en	549	22329304+28966	473	11009852+17705	50.46
100324523	ProphetAlerts.cc	ProphetAlerts		Just thoughts and cli	en	3556	397813137+3966	278	249450052+3742	100.56
100364439	????	yamamoto1975	???	????	ja	66	267767348+2645	77	154088683+2652	9.89
100441318	Sir Kumference	sirkumference		Knight of the very ro	en	25	379555085+2523	283	327243160+2878	5.68
100511648	sergiovp	sergiovelezp		Economist	en	437	320510943+3525	86	155996033+3354	10.56
100592354	Legacy Trades	Legacy_Trades	Here, there & e	Legacy Investments	en	5672	393466499+3743	165	209907837+2799	150.54
100597465	#CPC can't hv m	CometsMum	Canada	Fair-minded curious	en	361	365859075+2576	127	398100516+4671	9.68
100708381	AfricanInvestor	AfricanInvestor		#African from #Ghan	en	759	19156797+39715	195	16731133+27402	13.69
100787644	Mike Finnegan	GoFinny	Indianapolis	I'm an IT Maven and	en	56	378200405+2459	101	25053299+62716	9.39

图 10.28　用户信息表 user_$aapl

$AAPL 股票用户信息表 user_$aapl 中，每个元组的属性依次为：用户 ID、用户名、显示昵称、所在地、用户描述、语言、粉丝数、粉丝集合、关注数、关注用户集合以及用户权重。其中粉丝集合以及关注用户集合分别通过 GET followers/ids 和 GET friends/ids API 查询得到，用户之间以“+”分隔。用户在 Twitter 平台上的网络关系用集合的方式表达十分不便，因此本系统还建立了用户关系表 relation_$aapl。该表主键为有关注关系的两个用户 ID，如图 10.29 所示。

id_V	id_U	influence_p	delaytime
14886375	17745360	0.1485	54.36
17745360	14886375	0.1268	32.89
52166809	17745360	0.0125	98.59
84444981	17745360	0.0752	67.16
3705681	17745360	0.1538	39.26
17880354	17745360	0.0059	100.59
15368728	17745360	0.1129	23.59
21059648	17745360	0.0698	198.26
17745360	21059648	0.1026	86.29
18341423	17745360	0.0359	97.29
63061242	17745360	0.1632	46.68
14115408	17745360	0.0965	203.68
27922928	17745360	0.0635	168.39

图 10.29　用户关系表 relation_$aapl

遍历微博信息表$aapl，统计每个微博用户发布的关于苹果公司股票的原创微博 OT_1、对话微博 CT_1 与 CT_2、转发微博 RT_1 的数量。通过 10.4.2 节中的公式计算微博用户的 4 种特征值：主题参与度、原创度、非对话强度以及内容影响力，并通过加和计算对$AAPL 感兴趣的用户的权值 W（u），保存在数据库表 user_$aapl 的 weight 列中，如图 10.28 所示。

在计算用户影响联合概率时，通过提出的算法可以尽可能少地扫描微博样本集合。样本集合中的每条微博就是一个曾发出的行为，通过判断行为相似度，将输入的样本集合按照行为 id 排序，具有相同 id 的行为则按时间先后排序，排序需求很容易通过数据库提供的排序功能满足。本系统提出的计算用户影响联合概率的算法对于 Jaccard 系数静态模型和连续时间模型，只需要对行为集合做一次遍历即可，而 Jaccard 系数离散时间模型对于 $t_{v,u}$ 的限制，需要对训练集做两次遍历。计算过程中，用户 v 对用户 u 的行为影响概率储存在用户关系表 relation_$aapl 的 influence_p 列，而行为平均影响延迟时间储存在列 delaytime 中，如图 10.29 所示。用户 u 所受的影响联合概率可通

过用户关系表中的数据动态计算。

当分析完社交网络中的用户各属性后，就可以对$AAPL 股票价格涨跌趋势进行预测。依然以对$AAPL 股票持看涨态度在社交网络中的传播为例，在 2011 年 11 月 15 日上午 9:30 股市开盘时，对 Twitter 平台上用户发布的微博内容进行实时抓取，user_$aapl 表中有 Fortune Magazine 等 53 个用户节点发表了看涨的情感态度，将用户加入到看涨用户集合 BullSet 中。通过改进的广度优先遍历算法，分析这些用户的看涨态度在社交网络中的传播情况。以这 53 名用户为源头，广度优先遍历至 11 层时，新增加的看涨用户发出看涨态度的预测时间接近收盘时间 16:00，算法运行结束。进而以相同的方式预测看跌情感在网络中的传播情况，最终预测的看涨用户集合中的用户数量以及看跌用户集合中的用户数量如图 10.30 所示。

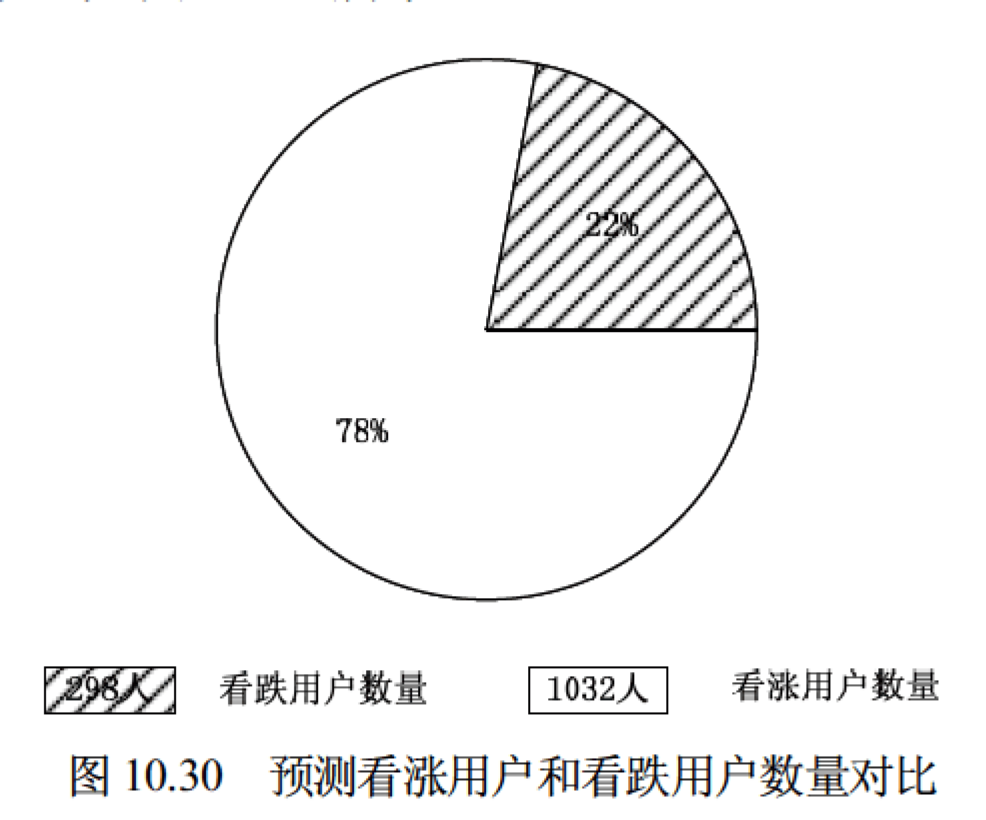


图 10.30　预测看涨用户和看跌用户数量对比

预测结果中有 1032 名看涨用户，有 298 名看跌用户，根据用户行为受情感支配理论，看涨用户的股市交易操作将直接影响股票价格的轻微上扬。而实际上，苹果股票的开盘价为$380.80，收盘价为$388.83，在 2011 年 11 月 15 日交易日内，苹果公司的股票上涨，而基于社交网络中情感信息传播预测模型预测苹果公司股票在本交易日内同样是上涨，这是一次成功的预测。

基于微博的股票市场预测系统界面如图 10.31 所示。

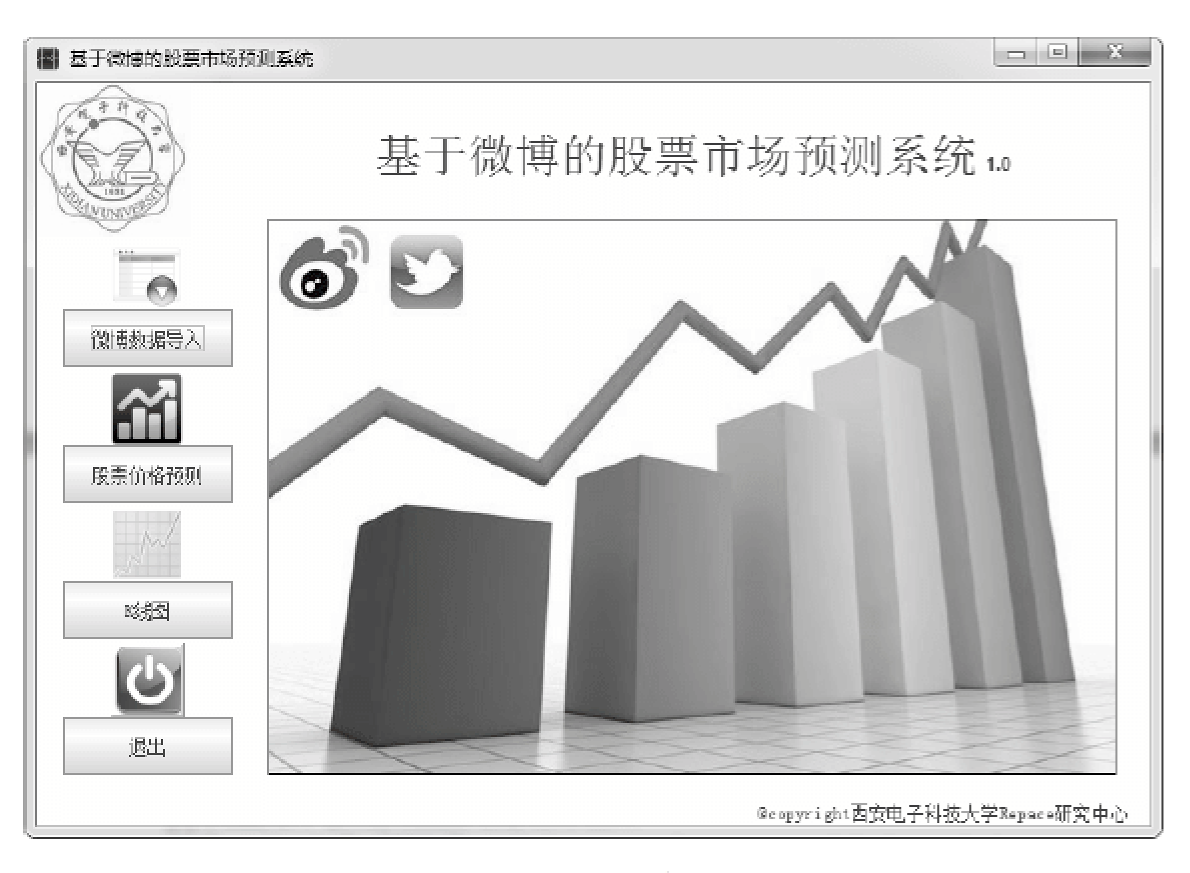


图 10.31　基于微博的股票市场预测系统主界面

从主界面可以看出，本系统为用户提供了三项基本功能：微博数据导入、股票价格预测、K 线图查询。在微博数据导入部分，系统可以显示需要查看的关于某支股票的 Tweets 消息；股票价

格预测部分，用户可以输入需要预测的股票代码和预测时间，系统会返回给用户一个预测结果；K 线图查询部分，根据股票代码用户可以查询某支股票的 K 线图。

在微博数据导入部分，用户可以输入需要导入微博的股票代码和时间，系统从数据库中将查询并返回给用户结果，包括每条微博的发布时间、作者和微博内容。如图 10.32 所示，系统查询了自 2011 年 8 月 1 日至 2011 年 12 月 21 日时间段内，Twitter 用户发表的内容包含$AAPL 的所有微博。



图 10.32　$AAPL 股票相关微博导入结果

股票价格预测时，用户需要输入预测的股票代码和预测时间，系统根据上文介绍的基于微博情感信息传播的股票价格预测算法，给出预测结果。如图 10.33 所示，用户输入股票代码 AAPL 和预测时间“2011-10-24”，系统将给出当天的 K 线图和预测结果，从图 10.33 中可以看出 2011-10-24 当天开盘价是$369.18，收盘价是$405.77，系统根据看涨和看跌的人数预测出看涨率为 1.19346，看跌率为 0.630899，预测结果为看涨，符合股市当前价格走势。所以本系统预测结果正确。



图 10.33　$AAPL 股票价格预测结果

最后，用户还可以查看某支股票的 K 线走势图，例如输入 AAPL，系统将根据抓取到$AAPL 股票历史的开盘价、收盘价、最高价、最低价，生成相应的 K 线图，用户可以通过 K 线图来判断预测结果的准确性，如图 10.34 所示。



图 10.34　$AAPL 股票价格 K 线图

另外，作者通过预测$AAPL 股票在 2011 年 10 月 24 日到 2011 年 10 月 18 日这 5 个交易日内的涨跌趋势，来验证该预测系统的有效性，得到的预测结果如表 10.12 所示.

表 10.12　$AAPL 股票涨跌趋势预测结果

预测时间	开盘价（$）	收盘价（$）	看涨用户数	看跌用户数	预测结果
2011-10-24	369.18	405.77	873	195	股价涨
2011-10-25	405.03	397.77	1132	1098	股价涨
2011-10-26	401.76	400.6	367	942	股价跌
2011-10-27	407.54	404.69	301	931	股价跌
2011-10-28	403	404.95	673	494	股价涨

通过表 10.12 并结合图 10.34 $AAPL 股票 K 线图可以看出，预测$AAPL 股票在 5 个交易日内的涨跌趋势中，只有 2011-10-25 一次预测错误，所以，在以上 5 个工作日内预测正确率均为 80%。

10.5 本章小结

本章作为大数据的应用篇，以 Twitter 海量数据为背景，介绍了一种基于 Twitter 微博的股票市场预测系统，并给出了该系统的详细设计与实现方案。首先，需要从 Twitter 社交网络中采集海

量用户信息和微博信息，然后，对这些海量数据进行预处理，完成微博情感分析和用户权威值的计算。再计算出社交网络中用户之间的影响关系和受到的联合影响概率，根据微博情感信息传播预测模型，统计出未来一段时间网络中对股票持看涨和看跌的用户数量。最后，预测出股票价格的涨跌趋势。通过实现表明，本系统在处理海量数据方面具有很高的效率，在预测股票价格涨跌时也能达到令用户满意的准确度。

第 11 章
基于内容的海量视频检索系统

在安防领域不断发展的前提下，视频监控成为了研究与应用的热点领域。本章主要介绍一个基于内容的海量视频检索系统，该系统运用 MapReduce 对视频中运动对象提取的方法进行了改进；使用 HBase 进行系统中相关数据的存储；采用一种新型的方法对检测到的运动对象进行行为识别，并创新性地利用规则组合的方式对复杂行为进行定义与检索。本章首先对该系统的功能性需求与非功能性需求进行了描述，深入分析并给出了系统的主要业务流程以及系统的整体架构图；然后对系统中涉及的各种相关技术进行简要的介绍；最后详细介绍系统中各个模块的设计与实现。

11.1 应用背景

智能视频监控技术是在传统视频监控技术的基础上发展起来的。传统的视频监控系统技术发展经历了三个时代：模拟时代、数字时代和网络时代。目前智能视频监控技术主要包括自动从视频图像序列中进行运动对象的提取、描述、跟踪和行为识别等功能。智能视频监控技术借助计算机强大的数据处理功能，对海量视频数据进行高速的分析，过滤掉用户不关心的信息，仅对用户提供有用的关键信息。智能监控视频技术的最终目的就是要使计算机能够分析、描述和理解视频画面中的内容，这其中涉及计算机视觉、图像视频处理以及人工智能等众多领域的核心技术。

目前，国内外对于智能视频监控技术的研究都有了很大的成就。其中，QBIC（Query By Image Content）系统是由 IBM Almaden 研究中心开发的，是一个典型的基于内容的视频检索，是最早的视频检索系统，检索条件包括关键字和视频的颜色、纹理、形状等特征。在 QBIC 系统中，提供三种查询方式：用户可以指定所需视频的主要颜色；用户可以指定所需视频中相似像素出现的机率；用户可以画出一个形状，按此形状进行查询。

IBM S3（Smart Surveillance System）是 IBM 公司 2009 年开发的一款智能化视频监控系统，是一款基于事件的检索系统，包含实时报警以及历史事件检索两部分。S3 是基于行为轨迹分析的先进的视频图像分析系统，能够实时分析视频信号，实时提取监控视频的行为轨迹和对象属性，发生异常产生报警信息，并以标准协议格式通过网络传送，提供实时报警显示、事件查询和历史时间统计分析等先进的功能。在视频检索方面，S3 是基于事件的检索，可以基于以下条件进行检

索：持续时间、目标类型、目标大小、目标颜色、目标在场景中出现的位置等。

虽然国外智能视频监控技术的研究要早于国内，但国内的许多高等院校和研究机构在智能视频监控领域都投入了大量的研究精力，其中包括上海交通大学、清华大学、华中科技大学和中国科学院自动化研究所等。而国内也有不少企业开始着手研究智能视频检索系统，其中关注度较高的是深圳久凌公司研制的“警用视频检索系统”。该系统首先使用运动分割方法提取运动的目标，而后这些运动目标的基本特征作为元数据被提取出来，存入数据库。在整个检索过程中，系统会将提取检索输入的特征，请求比对数据库已经索引号的目标特征，而不用重新处理视频。最后，拥有足够高相关度的视频目标快照将被作为结果显示出来。同时可以通过提供目标特征（人车类别、颜色、高度、方向、速度等）、一副样照或者素描图来请求搜索。该系统支持基于事件、目标分类或者样本检索等模糊查询检索功能。

通过智能视频监控系统可以大大降低人工的观察时间，提高系统的数据分析能力与相关视频检索的能力。本章将要介绍的“海量监控视频检索系统”就是通过 MapReduce 框架不断地对一个或者多个摄像头产生的视频以及已经存在的视频进行处理，提取出视频中出现的运动目标，并对目标的行为以及外形特征进行分析提取，将处理结果存入 HBase 数据库中。当用户在浏览器中上传需要查询的人物图像或者输入指定的时间、地点、行为等信息后单击查询，服务器将会首先对上传的图像进行处理，提取出图像中人物的颜色、纹理等特征，然后将提取出的特征值与数据库中所存储的各个对象的特征值进行比较，筛选出最为接近的几个结果，最后再通过时间、地点、行为等查询条件最终确定输出结果。为了尽可能提高准确性，输出的结果是与查询条件最为接近的一组视频数据。这一组视频数据为经过处理后仅具有用户感兴趣信息的视频片段，用户可以仅仅浏览输出的视频片段，也可以通过视频片段来查看原始的视频数据。通过该系统对监控视频进行处理与检索，可以大大提高数据的分析效率、检索效率以及检索精度。

11.2 需求分析与总体设计

11.2.1　功能需求

根据 11.1 节的描述，系统的功能需求已经非常清楚，图 11.1 给出了该系统的核心用例图，明确了系统的主要功能需求。

从图 11.1 中可以看出，海量视频检索系统有 4 个核心用例：系统对摄像头采集到的监控视频进行视频预处理，将运动对象存入数据库的同时对运动对象进行行为识别，用户根据需要进行视频检索，当系统中所存在的行为无法满足用户的查询需求时，用户可以通过规则创建来自定义活动。

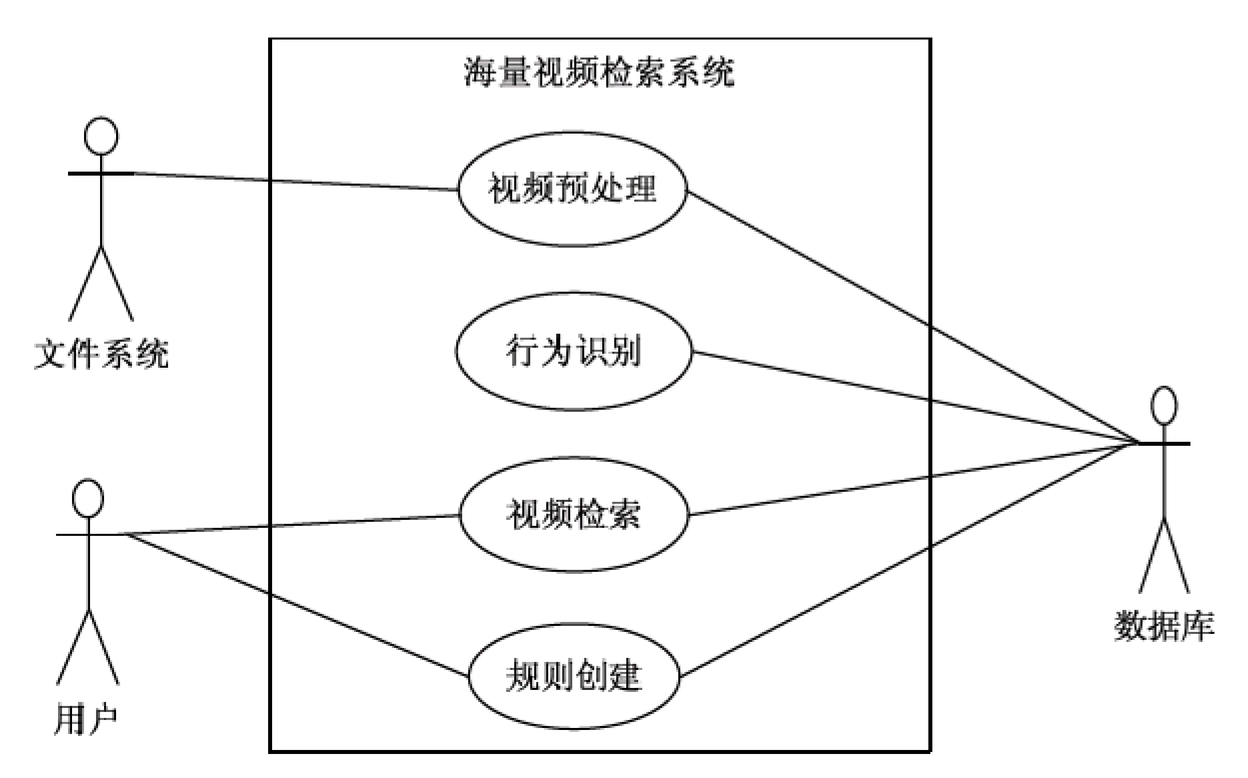


图 11.1 海量视频检索系统核心用例图

用户与系统的交互仅存在于视频检索的用例中。将监控视频文件的存储路径告知系统后，系统将会自动处理指定路径下的所有视频文件，其中包括视频的预处理、行为识别这两个用例。这两个用例都是系统的内部用例，由视频文件触发，不需要任何的人工干预，每个用例的处理结果都将存储于数据库中。用户对于视频的检索就是输入查询条件从数据库中返回结果集的过程。

1. 视频预处理

在该用例中，系统需要对指定路径（视频采集卡存储监控视频的路径）下的每一个视频文件进行处理。针对每一个视频，系统首先对视频进行分析，提取出视频中的运动对象后存入数据库中，然后对每一个运动对象的颜色、轮廓、纹理等特征值进行提取并存入数据库中，最后需要对视频的属性信息，例如时间、帧率等进行提取入库。

2. 行为识别

在该用例中，系统需要对数据库中已经存在的运动对象进行行为识别，并按运动对象的不同行为生成相对应的视频片段，便于检索使用。针对每一个运动对象，系统首先利用星形骨架方法对运动对象进行处理，然后提取不同的角度特征值并与已经制定的规则进行对比，最后判断出对象的行为类型后入库，并将该行为所对应的行为片段进行抽取生成。

3. 视频检索

在该用例中，用户可以对自己需要的视频进行检索与播放。首先用户在系统界面中输入一种或几种查询条件（时间、地点、人物、行为），然后系统将会在数据库中按照一定的次序查询是否存在满足条件的视频，最后如果存在满足条件的视频将会输出视频列表（用户双击视频名即可对视频进行播放），如果不存在满足条件的视频将出现提示信息。

4. 规则创建

该用例中，注册用户可以对视频搜索系统的规则库进行充实，创建新的规则。通过将组成新规则的一系列图片，按照其中对象行为发生的先后顺序排好序，并放于某一文件夹下，在系统相应界面上，输入正确的对应的路径，而创建新的规则。

在核心用例图的指导下，设计出每个用例的详细交互过程，并以表格形式进行详细的用例描

述，包括视频预处理（如表 11.1 所示）、行为识别（如表 11.2 所示）、视频检索（如表 11.3 所示）、规则创建（如表 11.4 所示）。

表 11.1　视频预处理详细用例表格

用例编号	UC-1	
用例名称	视频预处理	
参与者	海量视频检索系统管理员	
描述	该用例描述了海量视频检索系统处理指定的文件路径下的视频文件的过程	
用例典型事件流	参与者动作	系统响应
	Step1：输入视频文件存放路径	
		Step2：查找到指定路径下的视频文件
		Step3：对单一视频进行运动对象提取
		Step4：对运动对象的特征值进行提取
		Step5：将处理的全部结果存入数据库中
		Step6：若文件夹中存在未处理的视频文件，跳转至 Step2
可选事件流	Step2，用户输入的路径下不存在任何视频文件，向用户返回“指定文件路径有误”提示信息。 Step6，经过检查，若指定路径下的文件均已处理完毕，则系统需要过 10 分钟后再次对指定路径下的文件进行检测，若连续两次检测均为出现新文件，则向用户返回“监控停止运行”的提示信息	
前置条件	指定路径下已经存放有至少一个视频文件	
后置条件	系统可以立即对数据库中的信息进行查询	
假设	无	

表 11.2　行为识别详细用例表格

用例编号	UC-2
用例名称	行为识别
参与者	系统自身
描述	该用例描述了将数据库中存在的运动对象处理，得到对象的特征值，然后与规则库中的规则进行匹配，识别出所对应的行为的过程
用例典型事件流	系统响应
	Step1：　有新的运动对象可处理
	Step2：提取运动对象并进行二值化处理
	Step3：提取每帧图片轮廓特征
	Step4：利用星型骨架方法获取每帧图片的行为特征
	Step5：采用最长字符串匹配方法，将视频中每帧图片的行为特征值与规则库中的规则特征进行匹配，识别出视频中包含的行为
	Step6：将视频中识别出的行为以及对应的帧号存入数据库中，以便于下一步搜索
可选事件流	Step1，当系统中无运动对象可处理时，系统不进行行为识别而处于等待状态，直到新的运动对象到达，再触发系统进行行为识别。 Step3，当轮廓模糊或者没有对象轮廓可提取时，则跳过当前帧的轮廓提取，进行下一帧图片的轮廓提取。 在关闭系统时，系统自动保存目前处理的运动对象的序列号，在下次启动时，自动按照上次未处理完的地方继续处理
前置条件	无
后置条件	将识别出的行为以及对应的帧号存入数据库中，以便于用户输入行为条件进行匹配搜索
假设	无

表 11.3　视频检索详细用例表格

用例编号	UC-3	
用例名称	检索对象视频	
参与者	用户	
描述	该用例描述了普通用户或注册用户检索视频的过程。用户上传一张对象图像，并选择时间段、地点、对象行为，系统通过对上述条件进行检索返回目标时间、地点、执行特定行为的目标对象视频，用户浏览检索结果	
用例典型事件流	参与者动作	系统响应
	Step1：上传一张对象图像	
	Step2：选择事件发生时间	
	Step3：选择事件发生地点	
	Step4：选择对象行为	
		Step5：确认操作成功
		Step6：时间检索
用例典型事件流		Step7：地点检索
		Step8：图像分析
		Step9：基于内容的图像检索
		Step10：对象行为检索
		Step11：向用户返回相关视频
	Step12：浏览检索结果	
可选事件流	Step5，用户上传图像时可能由于网络超时等原因失败，向用户返回“图像上传失败”提示信息。 Step1、Step2、Step3、Step4 这几个步骤都是可选的，即所有单项查询条件可以为空，但是这 4 个步骤必须存在一个步骤，也就是必须有一个查询条件不为空。 Step11，经过检索，发现没有相关对象视频命中，向用户返回“未找到相关视频”提示信息。 在 Step1 之后，Step11 之前，用户可以关闭应用程序或检索页面，系统自动停止检索	
前置条件	无	
后置条件	用户可以对检索结果进行浏览	
假设	无	

表 11.4　规则创建详细用例表格

用例编号	UC-4	
用例名称	规则创建	
参与者	用户	
描述	该用例描述了用户通过输入一张图片或者一组图片创建新规则的过程。用户输入一张图片或者一组图片的路径以及要创建规则的规则名，系统通过用户输入的路径获取到图片，对每张图片进行处理，获取到规则特征值，然后将规则的特征值以及对应的规则名称存入规则库，完成规则的创建过程	
用例典型事件流	参与者动作	系统响应
	Step1：输入规则图像路径及规则名称	
		Step2：确认路径有效
		Step3：图像分析处理，得到规则特征值
		Step4：规则特征值及规则名称入库
		Step5：向用户反馈创建结果（创建成功/创建失败）
	Step6：继续下一步操作	

（续表）

可选事件流	Step1，用户上传图像时可能由于输入错误等原因失败，向用户返回“路径输入有误”提示信息。 Step3，在图片处理过程中，由于图片模糊或者图片无对象造成无法正常获取特征值，向用户返回“图片不存在”提示信息。 Step4，在存入规则库的过程中，由于某些意外原因无法正常存入规则库，向用户返回“入库异常”提示信息。 在 Step1 之后，Step5 之前，用户可以关闭应用程序，系统自动停止规则创建过程
前置条件	普通用户和注册用户均可输入路径名及新规则的名称
后置条件	用户可以继续创建规则或进行其他操作
假设	无

11.2.2　非功能需求

除了上述功能性需求，海量视频检索系统需要满足的非功能性需求主要包括性能、可扩展性和可用性等方面。

性能方面，海量视频检索系统向用户提供信息检索业务，用户可以通过网页对所需信息进行查询。根据用户调查，用户能够容忍的等待时间最大为 3.0 秒，最优的等待时间为 1.0 秒之内。因此，针对数据量较大的特点，当查询时间超过 1.0 秒时，需要适当地给出一些加载提示让用户知道查询正在进行中，并且可以很快看到结果。

可扩展性是保证系统竞争力的关键。海量视频检索系统依赖于监控摄像头所拍摄的视频，理论上讲，越多的监控摄像头拍摄越多的视频，系统检索到有用信息的准确率就会越高。面对不断增加的摄像头，系统的平滑扩展就显得十分重要。因此，海量视频检索系统应该能够方便地通过增加系统所需处理的文件夹的方式实现扩展，即将新增加的摄像头视频存储文件夹的地址告知系统，以此来增加系统对新增摄像头下视频的处理与查询。

最后，对于原始视频数据应该保证数据安全，在没有管理员权限的情况下不能随意删除、更改原始视频数据。

11.2.3　核心业务处理流程

在对系统用例进行深入分析的基础上，本小节给出了视频预处理、行为识别、数据检索和规则创建的详细业务处理流程。下面将分别予以说明。

1. 视频预处理处理流程

随着监控视频的流行，每天都会产生海量的视频文件，但是用户无法直接从原始视频中得到有用信息。为了得到相关信息，如果用户对视频进行人工浏览，一方面会浪费大量的时间与人力，另一方面由于视觉疲劳的缘故用户无法找到所有的有用信息。另外，视频中我们主要关注的是运

动的对象，同一对象很可能持续出现，这也降低了人工浏览的效率。那么，现在需要解决的问题是，如何在不需要人工参与的情况下仅仅提取运动的对象，以及如何区分运动对象。

在海量视频检索系统的视频预处理中解决了以上的所有问题。视频预处理的详细处理流程如图 11.2 所示。

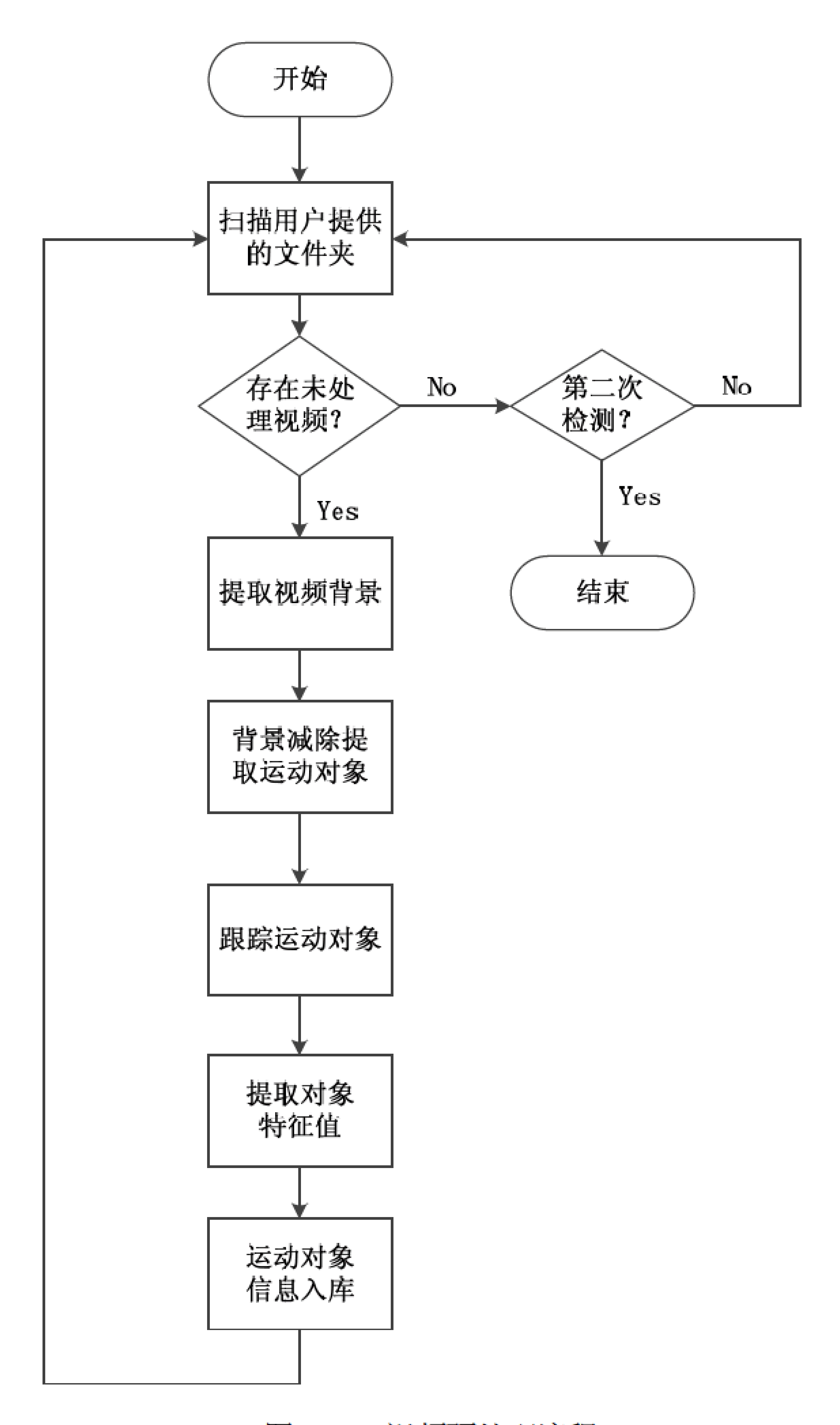


图 11.2　视频预处理流程

由于监控摄像头一般为 24 小时全天候工作，因此将会不间断地产生待处理的视频文件。当扫描发现文件夹中目前不存在未处理的视频文件时，系统将会在 10 分钟后再次对该文件夹进行扫描，当两次扫描文件夹均未发现新的视频文件时，则可以判定监控摄像头停止工作，系统将会结束视频的处理模块。如果持续检测到新文件的产生，则该用例模块将会一直循环执行下去。

由于监控摄像头一般情况下均是固定放置，因此每一个监控摄像头拍摄的视频中背景一直是固定不变的。针对监控视频的这一特点，系统采用“背景减除法”对视频的运动对象进行提取。首先，系统需要提取出视频的背景；然后通过混合高斯模型将背景图片与视频帧进行对比，提取

出运动对象；接下来利用 Mean shift 算法进行运动对象的跟踪；最后提取运动对象的颜色、形状、纹理等特征值并将每一个运动对象的所有信息存入到后台数据库中。

2. 行为识别处理流程

行为识别的过程是将视频的所有对象发生的所有动作，通过提取行为特征值，并将行为特征值与规则库中的规则进行匹配，识别出所有对象所发生的行为，以及对应的帧号，最后将识别出的对象的行为以及帧号存入数据库，具体流程如图 11.3 所示。

3. 视频检索处理流程

检索过程的主要流程如图 11.4 所示。首先提取用户选择的事件时间、地点信息，将这些信息与数据库存储的相关信息进行比对，得到一个中间结果集；接着提取图片的特征值，然后与图像数据库中的所有图片特征值进行检索比较，由于采用基于内容的图像检索（Content-Based Image Retrieval， CBIR）技术；最后显示给用户的相关视频准确与否取决于用户上传图像和数据库中的图像之间的内容相似度。所谓图像内容，一般指图像本身包含的底层特征信息，例如颜色、纹理、形状和空间关系等可视特征。为此，系统预先提取了数据库中所有图像的内容特征，存入数据库并进行索引。这是一个特征值相似度计算的过程：当系统获取到用户上传的图片时，同样会首先提取该图片的特征值，然后按照一定的 CBIR 相似度对比算法进行相似度计算，并对最终的相似度排序。

最后再提取对象执行的行为信息，将它与数据库中存储的行为信息进行比对，返回特定时间、特定地点、执行特定动作的目标人物视频。检索结果会按照与上传的图片特征相似度由高到低的排序返回给用户，为了过滤相似度较低的图片条目，系统设置了一个相似度阈值，当相似度低于该阈值时，系统认为这些条目与用户上传的样本图片并不相似，而相似度高于该阈值的所有图片构成命中结果集。需要读者明确的一点是，一个对象在一个时间段内执行某行为的视频片段可能是多个。例如，张三在中心广场某日 14:00~14:30 这半小时的时间里，他先跑了 10 分钟，接着走了 10 分钟，接着又跑了 10 分钟。如果用户上传张三的图片，时间选定为 14:00~14:30，地点选定为中心广场，对象执行行为选定为跑，则最终会给用户返回 14:00~14:10 以及 14:20~14:30 这两个时间段内的视频。

用户在这些相关结果中选择与自己上传的人物对象图片相似的视频片段进行观看，系统在提供执行特定行为的视频片段之外，用户还可以根据需要观看原始视频数据作为参考。当用户得到满意的结果后可以接着对自己感兴趣的内容进行查询。如果系统经检索后，数据库中图片特征值与用户上传图片的特征值无法匹配，即该图片与数据库中所有图片的相似度都低于相似度阈值时，系统则提示用户没有找到结果，建议用户重新拍摄图片并上传。

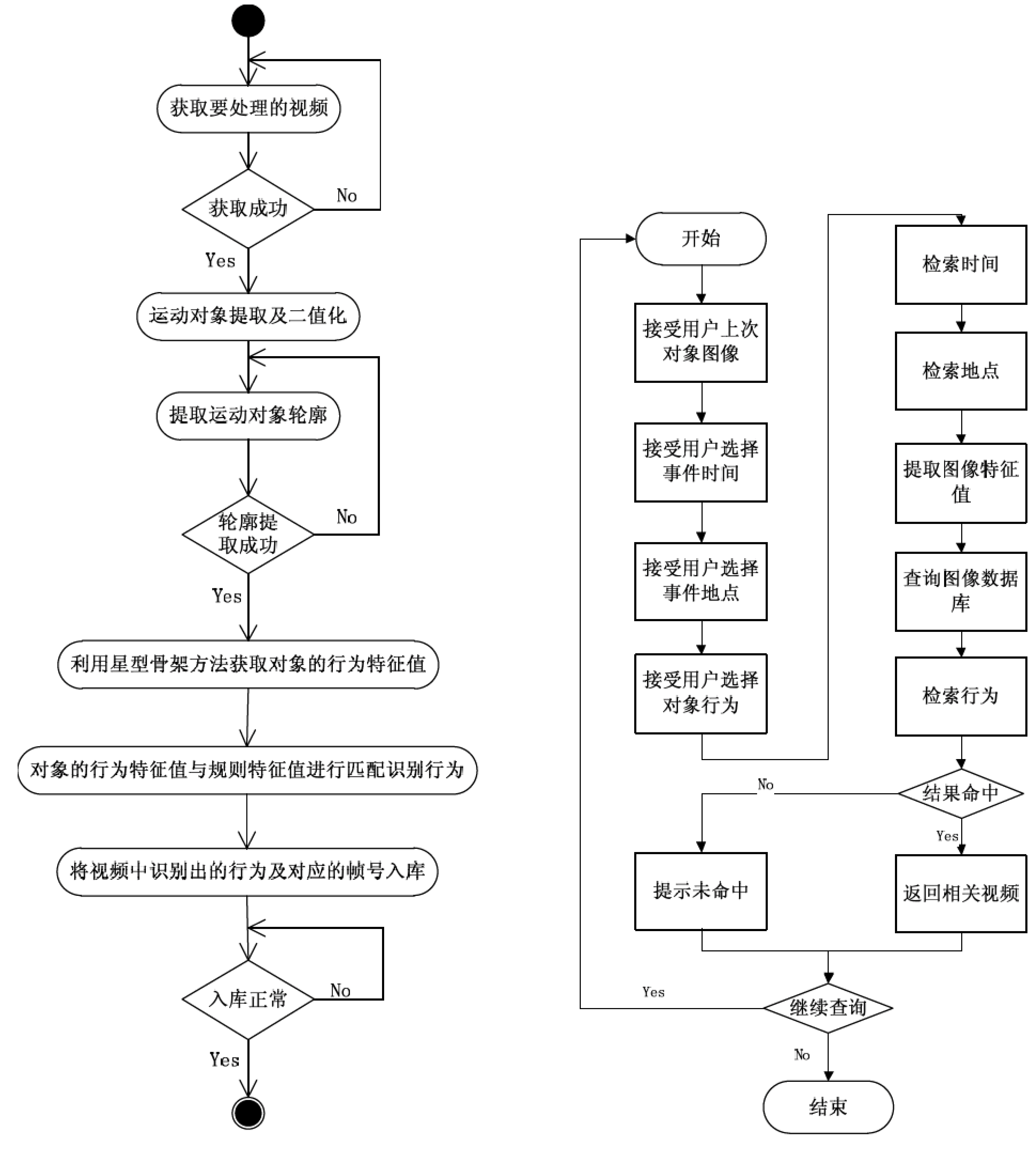


图 11.3　行为识别处理流程　　　　图 11.4　视频检索处理流程

4. 规则创建处理流程

需要创建一个新的规则时，用户只要单击页面中的“创建规则”按钮，根据提示输入相关信息就可以创建规则。其中已经存在的规则不可以重复创建，新的规则用户通过自己输入组成规则的图片的路径进行创建。组成新规则的图片必须是按照图片中对象行为发生的先后顺序组合。组成新规则的图片可以是一组对象的行为序列，也可以是几组对象的行为序列。对于几组对象的行为序列，采取的策略是，将每组对象求出的规则特征值，求平均值后得到最后的规则特征值，生成新的规则特征值需要经过验证后才可以存入规则库中使用。图 11.5 为规则创建的处理流程图。

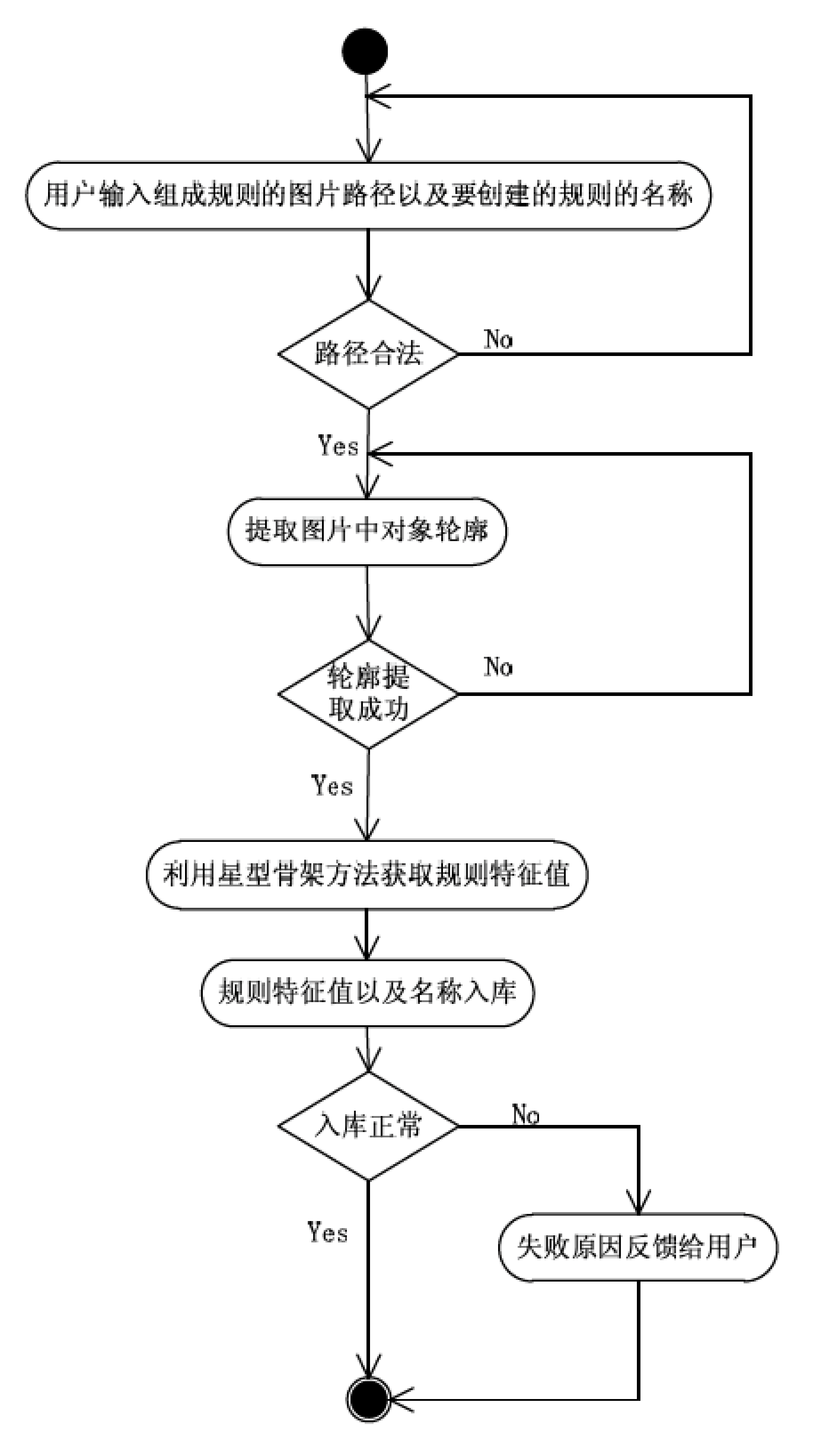


图 11.5　规则创建处理流程

11.2.4　总体设计

图 11.6 为基于内容的海量监控视频检索系统的上下文数据流图，该图确定了基于内容的海量监控视频检索系统全局的系统边界，显示了该系统和系统外部实体（如用户和其他系统）之间的边界和接口。

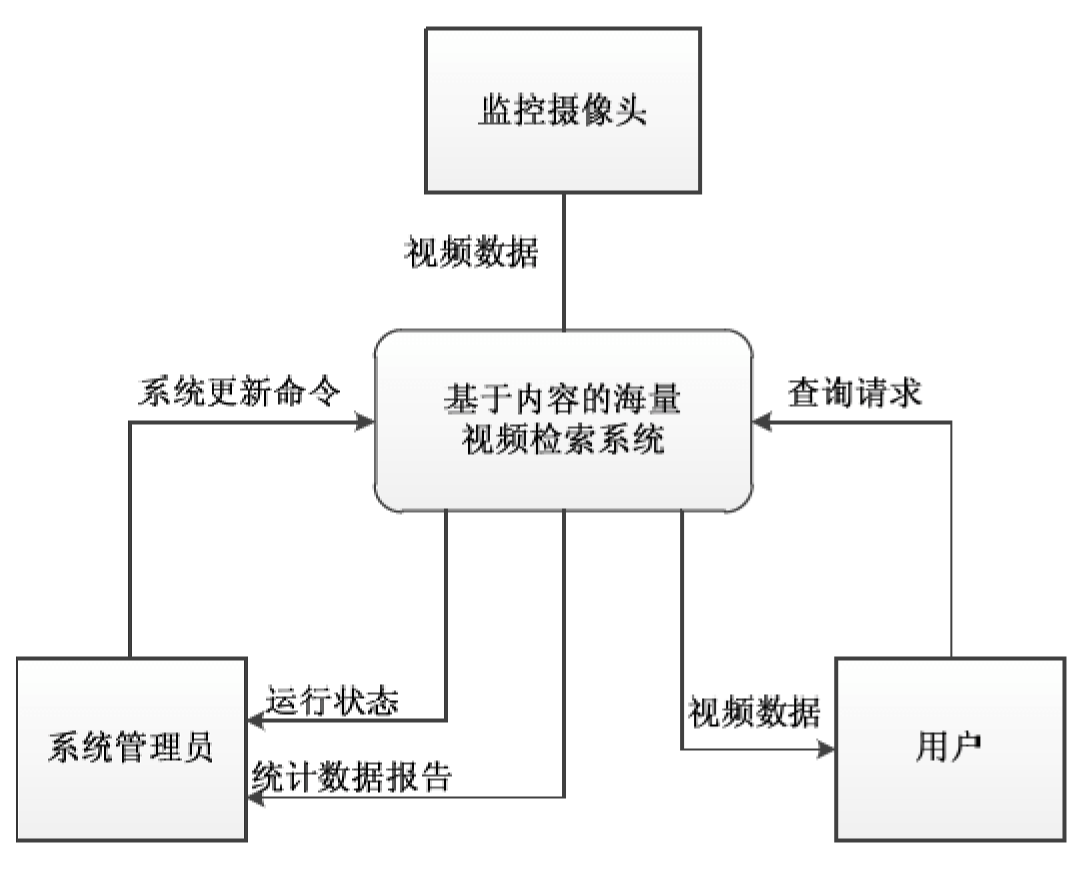


图 11.6　海量监控视频检索系统的上下文数据流图

用户是系统交互的核心对象，具有提交查询请求的权限，能够向系统提交对象图像、事件发生时间、地点以及该对象执行的行为进行检索。系统响应用户查询请求，并返回查询到的相关视频片段。

系统管理员角色会管理基于内容的海量监控视频检索系统，并对系统发出更新命令，这里的更新可能包括系统本身功能的更新和数据的更新两种。另外，管理员角色还会接受来自系统的运行状态报告以及各种统计数据报告。

基于内容的海量监控视频检索系统的视频数据来源于与电脑直接连接的监控摄像头，基于内容的海量监控视频检索系统会定期读取存储在电脑中特定文件夹中的原始视频数据，并进行数据更新，保证了用户查询结果的实时性、准确性以及系统数据的完备性。

基于上述详细的需求分析，对系统进行了总体设计，图 11.7 给出了基于内容的海量监控视频检索系统的总体架构图，从图中可知图像百科系统自下而上可以分为 7 个主要层次。

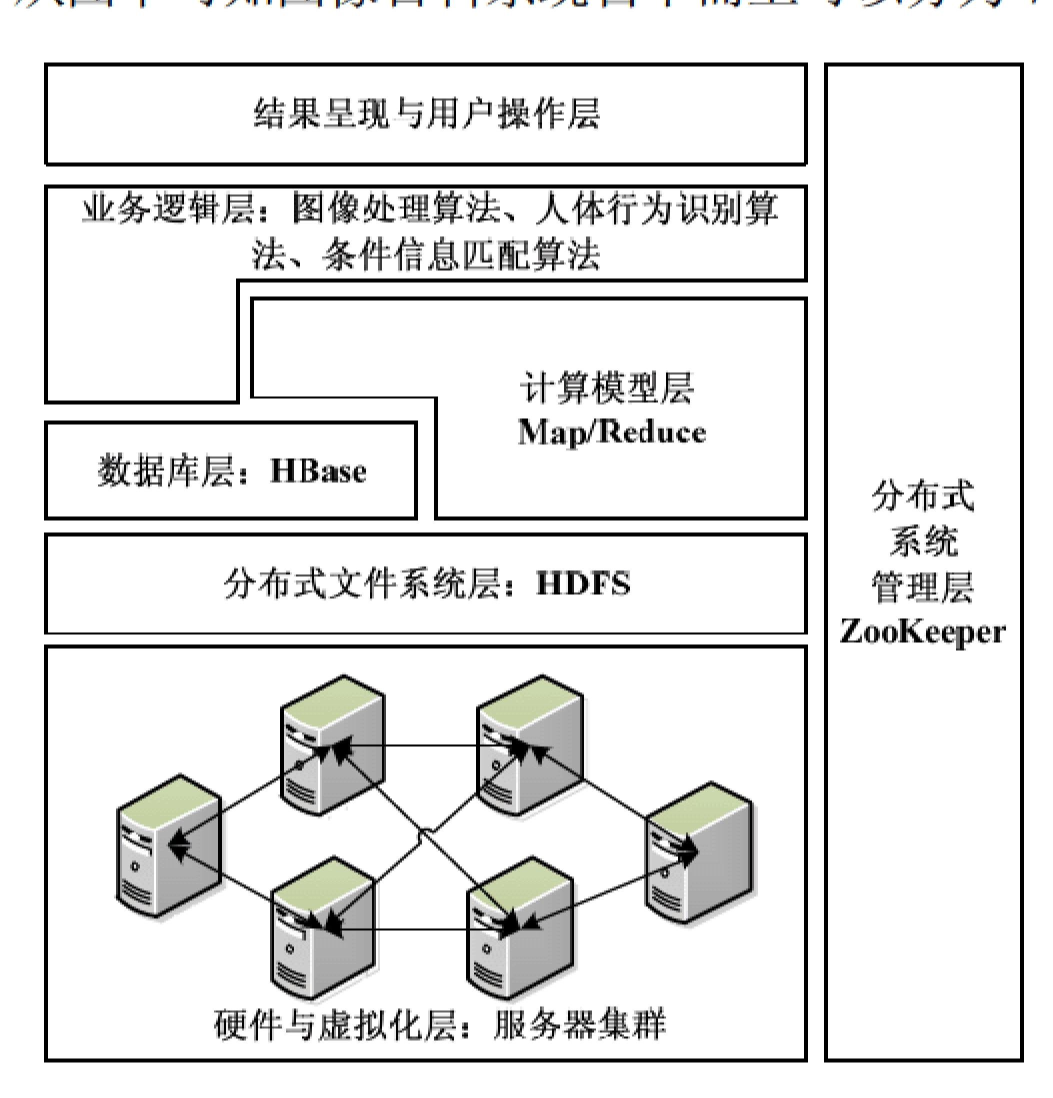


图 11.7　海量监控视频检索系统总体架构

1. 硬件与虚拟化层：服务器集群

该层的主要作用是为基于内容的海量视频检索系统提供底层的硬件与操作系统的基础环境支持。基于内容的海量视频检索系统底层采用云计算技术，整个计算环境构建在一个采用了虚拟化技术的 PC 机服务器集群之上。

2. 分布式文件系统层

该层的主要作用是为基于内容的海量视频检索系统提供底层的存储平台。分布式文件系统层位于整个系统的底层，它向计算模型层、数据块层以及 Web 访问提供统一的访存接口，其他模块通过接口的调用可以很方便地在分布式文件系统中存取数据。同时，该层的副本策略、负载均衡等机制也保证了存储平台的可用性和可靠性，为整个系统提供稳定的可依赖的存储空间。

3. 数据库层

该层的主要作用是为基于内容的海量视频检索系统提供数据存储支持，能够向业务逻辑层和计算模型层提供数据持久化功能，并通过合理的主键设置和索引机制，向上层提供高效的数据访问能力。基于内容的海量视频检索系统采用 HBase 作为数据库，能够满足系统高并发、高可扩展性的需求。基于内容的海量视频检索系统的图像特征、时间信息、地点信息、对象行为信息的数据都存储在 HBase 中。

4. 计算模型层

该层的主要作用是当基于内容的海量视频检索系统需要进行海量数据处理时，可以在大量普通配置的计算机上实现并行处理。对海量的原始视频进行预处理，包括将视频处理成海量对象图片，对图片特征值进行抽取及对人体行为特征进行识别时，这些海量数据的处理都要借助于 MapReduce 计算模型。MapReduce 计算模型封装了并行处理、容错处理、本地化计算、负载均衡等细节，还提供了简单而强大的接口。通过这个接口，可以把大数据量的计算自动地并发和分布执行，使之变得非常容易。这里需要注意的是，考虑到 MapReduce 分布计算模型并不适合于需要即时响应的计算过程。用户上传的图片与数据库中特征值进行匹配这一过程并不适用该计算模型。

5. 业务逻辑层

该层的主要作用是进行与图像处理、人体行为识别、时间和地点信息匹配相关的操作，调用下层的计算模型层进行数据处理，并通过数据库层实现对数据的读写操作。具体来说，该层提供图像特征抽取、图像匹配、人体行为识别及条件信息匹配、数据入库等业务。特征抽取是通过调用计算模型层的 MapReduce 分布计算模型，将对本地监控视频处理后获得的关键帧图片底层特征在各个计算节点抽取并存入数据库层中；人体行为识别是通过调用计算模型层的 MapReduce 分布计算模型，对从原始视频中提取出来的人体对象在各个节点使用人体行为识别算法进行处理，并将中间结果移交给系统其他模块，最后将识别出来的人体行为视频片段存入数据库层中；条件信息匹配则是从用户操作层获取用户提交的时间、地点、人体行为等特征，并与数据库层中的相关数据进行匹配。

6. 结果呈现与用户操作层

该层的主要作用是接受用户对视频库中的数据进行查询的操作请求，对用户身份和操作的合法性进行验证，并将请求转发到业务逻辑层。基于内容的海量视频检索系统支持浏览器网页方式（例如 IE、Firefox、Chrome 等）的请求发送与结果呈现。基于内容的海量视频检索系统使用 Web 服务器接受请求，并进行操作的验证与处理，通过动态页面生成技术向用户呈现合适的结果。

7. 分布式系统管理层

该层的主要作用是对整个系统的服务器集群进行管理，同时提供全局配置信息管理功能。ZooKeeper 的集群机器监控能够对集群中机器的变化做出快速响应，从而提供高可用性服务。在基于内容的海量视频检索系统中，部署在多个机器上的应用程序需要感知整个集群的状态，以及

某个机器的信息，并在机器状态变化时能触发自己的事件处理程序，所以在应用程序中都有一个集群管理器的实例，应用程序通过集群管理器访问/管理集群。集群间的公有配置信息需要集中管理，这一点通过 ZooKeeper 的配置管理功能实现。将公共的配置信息放到 ZooKeeper 服务器的特定目录下作为共享配置，一旦配置发生变化，每台应用服务器就会收到 ZooKeeper 的通知，获取最新配置信息。

11.3 相关技术简介

上一小节对海量监控视频检索系统的总体设计以及各用例的处理流程都做了详细的介绍。本小节将对系统实现中涉及的相关技术进行简要介绍。

11.3.1 MPEG-7 与 OpenCV 简介

1. MPEG-7 简介

多媒体内容描述接口（Multimedia Content Description Interface），简称为 MPEG-7，是运动图像专家组（Moving Picture Experts Group）提出的用来描述多媒体内容的标准，其目标是创建一种描述多媒体内容数据的标准，满足实时、非实时和推-拉应用的需求，以及支持数据管理的灵活性、数据资源的全球化和互操作性。这种描述能对信息的内涵进行某种程度上的解释，而且能被计算机或其他信息设备传递或访问。

MPEG-7 支持多种音频和视觉的描述，包括自由文本、N 维时空结构、统计信息、客观属性、主观属性、生产属性和组合信息。对于视觉信息，描述将包括颜色、视觉对象、纹理、草图、形状、体积、空间关系、运动及变形等。MPEG 并不针对应用标准化，但可利用应用来理解需求并评价技术，它不针对特定的应用领域，而是支持尽可能广泛的应用领域，就目前来看，适合应用 MPEG-7 的领域包括：基于内容的多媒体搜索（包括图像搜索、哼唱搜索、语音搜索等）、图像理解以及其他需要使用大量多媒体特征的应用。

MPEG-7 实质是对信息进行描述的标准。如图 11.8 是 MPEG-7 处理链的抽象示意图。它包括特征提取（Feature Extraction）、标准描述（Standard Description）和搜索引擎（Search Engine）。MPEG-7 的前端是特征提取，也就是说 MPEG-7 是建立在特征提取基础上的，它只对已经过处理得到的特征信息进行描述，而并不关心这些特征是如何得到的。MPEG-7 的后端是搜索引擎，它利用根据 MPEG-7 标准描述的特征内容来进行检索得到检索结果集，即搜索引擎是对 MPEG-7 描述内容的具体实现及应用。由此可知，MPEG-7 只是在特征提取和信息检索之间提供标准接口，本身并不直接参与这两者的实现，而只在基于内容的检索中起着连接两者并提供接口进行实现的作用。

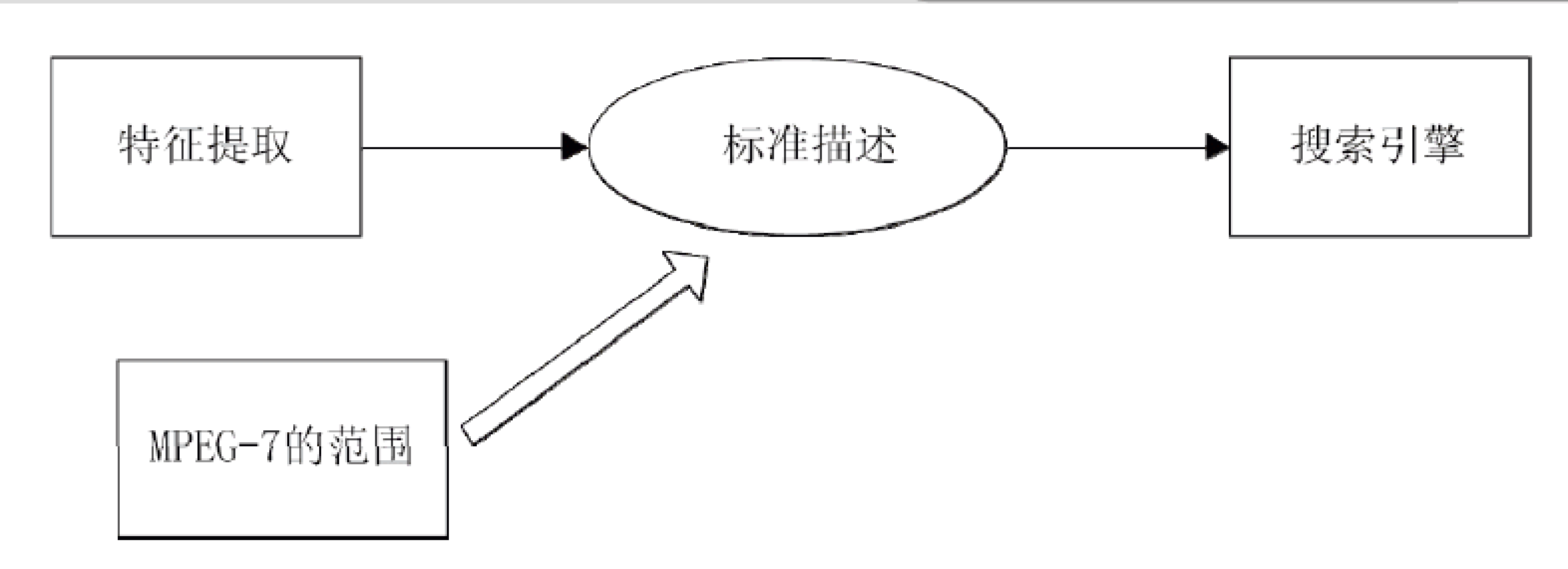


图 11.8　MPEG-7 处理链的抽象示意图

MPEG-7 可以处理的对象包括静止图像、图片、三维模型、语音、活动图像以及这些媒体的综合信息，对于不同的数据类型，MPEG-7 对应有不同的标准，即 D 和 DS，所以 MPEG-7 描述与被描述内容的表达无关。

MPEG-7 的构成要素包括：描述工具、描述定义语言以及系统工具。描述工具包括一组描述符 D（Descriptor）和描述方案 DS （Description Schemes）。描述符是指用来定义和表达实体某一方面特征的句法或语法，描述方案由一个或多个 D 和 DS 构成，DS 规定了它们相互关系的结构和语法。描述定义语言 DDL（Description Definition Language）是用来指定描述方案的一种语言，它是一种模式化语言，是对音视频数据建模结果的一种表征。DDL 语言是 MPEG-7 的一个核心部分，提供了基本的多媒体描述的方法，使用户能够创建自己的 DS 和 D。DDL 语言发展兼顾和吸收了现有的各种媒体描述语言的特点，其中对其影响最大的是 XML Schema 语言和资源描述框架（RDF）。DDL 可分为几个部分：

- XML Schema 结构语言部分，包括定义和声明以外的 XML 内容、简单类型定义、复杂类型定义、属性声明和元素声明；属性组定义、约束身份定义、组定义和符号定义：注释、通配符等。
- XML Schema 数据类型语言部分，提供了若干内置的数据原型和派生类型，以及用户自定义数据类型机制。
- 特殊扩展部分，对 XML Schema 的扩展主要表现在对数据类型的添加和扩展，增加了数组矩阵类和时间类。系统工具则用来支持多路描述、同步问题、传输机理、文件格式等。

2. OpenCV 简介

OpenCV（Open Source Computer Vision Library）是由 Intel 公司资助的开源的并且可跨平台的计算机视觉库，可以运行在 Linux、Windows 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和 C++类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。OpenCV 包括 300 多个 C/C++函数的跨平台的中、高层 API，它不依赖于其他的外部库，但是也可以使用某些外部库。OpenCV 1.x 版本中提供的主要是 C 语言的接口，在新发布 OpenCV 2.x 版本中提供了大量算法的 C++接口，下面将主要对 OpenCV 2.x 版本进行简要介绍。OpenCV 2.x 版本主要由以下几个模块组成。

- core：此模块内对 OpenCV 的基本数据结构进行了定义，其中包含矩阵类型 Mat（此类型代替了 OpenCV 1.x 版本中图像类型 IplImage 以及矩阵类型 CvMat）以及可以用于其他模

块的基本功能。

- imgproc：此模块专门用于图像处理，包含线性与非线性的图像滤波器、图像的几何变换、颜色空间转换、直方图等。本系统中对图像的平滑处理、边缘检测、轮廓提取、颜色直方图等基本图像操作函数均出于此模块。
- video：此模块主要用于视频分析，包含运动估计、背景减除、目标跟踪等。
- calib3d：此模块的主要功能是对元素的 3D 重建以及相机定标等。
- features2d：此模块用于对特征点进行检测、描述以及匹配。
- objdetect：此模块的主要功能为对象检测以及制作样本库（例如面部、眼睛、人、车等）。
- highgui：此模块可以容易地实现视频真捕捉、图像与视频的编码以及简单的界面。
- gpu：此模块实现了来自与不同 OpenCV 模块的 GPU 加速算法。
- ml：此模块为一个机器学习模块，提供了强大的机器学习类来实现数据的统计划分、聚类等功能。

以上即为组成 OpenCV 2.x 版本的主要模块，还有一些辅助性模块，在此不再赘述。表 11.5 介绍了本系统所用到的部分 OpenCV 库函数。

表 11.5　系统中用到的部分 OpenCV 库函数

函数名	函数功能
Canny 边缘提取函数 (CvCanny)	采用 Canny 算法寻找输入图片的边缘并在输出图像中标志这些边缘
FindCounters 轮廓提取函数 (cvFindCountours)	在二值图像中寻找轮廓，并返回轮廓的数目
Moment 形状提取函数 (cvMoments)	计算最高达三阶的空间和中心矩，并且将结果存在结构 moments 中
CreateHist 颜色提取函数 (CvHistogram)	此函数提取图片的颜色特征，创建一个指定尺寸的直方图，并返回创建的直方图的指针

11.3.2　运动对象提取

本系统中运动对象提取可以分为运动目标检测与运动目标跟踪两部分。运动目标检测可以从视频中检测到运动的对象，运动目标跟踪可以用来记录运动对象的运动轨迹等信息。下面将分别对两种技术进行介绍。

1. 运动目标检测

运动目标检测技术是智能监控系统的首要问题，与后续的目标跟踪、行为识别等有着密切的关系。运动目标检测就是从图像中将感兴趣区域从背景图像中提取出来，这是视频处理与编码的关键。换句话说，运动目标检测技术就是从视频图像序列中将运动变化的区域一帧一帧地检测出来，通过去除背景图像的影响从而找到感兴趣的目标或者前景。根据背景是否固定，运动目标检

测可以分为静止背景下的目标检测与活动背景下的目标检测。由于监控摄像头一般情况下是放置在一个固定的位置上，排除外界光线变化、摄像头轻微抖动等因素，监控视频中的目标检测是典型的静止背景下的目标检测。目前常用的静止背景下检测运动目标的算法主要分为三类：光流法、帧差法和背景减除法。

（1）光流法

光流法是一种以灰度梯度基本不变或亮度恒定的约束假设为基础的有效的目标检测方法。光流场是空间运动物体被观测面上的像素点运动产生的瞬间速度场，包含物体表面结构和动态行为的重要信息。这种方法的基本原理是：对图像中的每一个像素点都赋予一个速度矢量，从而形成一个图像光流场；在运动的某个特定时刻，将图像上的像素点投影到三维物体上的点；然后根据每个像素点的速度矢量特征来动态分析图像，如果图像中不存在运动目标，那么整个图像区域的光流矢量是连续变化的，如果存在目标相对图像背景运动，那么运动目标所形成的速度矢量将不同于邻域的背景速度矢量，从而检测出运动目标的具体位置和相关运动参数。

光流法不需要知道关于图像背景的任何信息，不仅适用于静止背景和动态背景下的视频运动目标检测，甚至可直接用于运动目标的跟踪。但是，在实际应用中，由于光源、阴影、噪声等原因，将会导致光流法得到的结果并不十分精确，并且光流法存在公式复杂、计算量大等缺点，因此实际应用中不常使用。

（2）帧差法

帧差法利用图像序列中相邻两帧或三帧图像中的对应像素值相减，然后取差值图像进行阈值化处理，提取出图像中的运动区域，基本过程如图 11.9 所示。

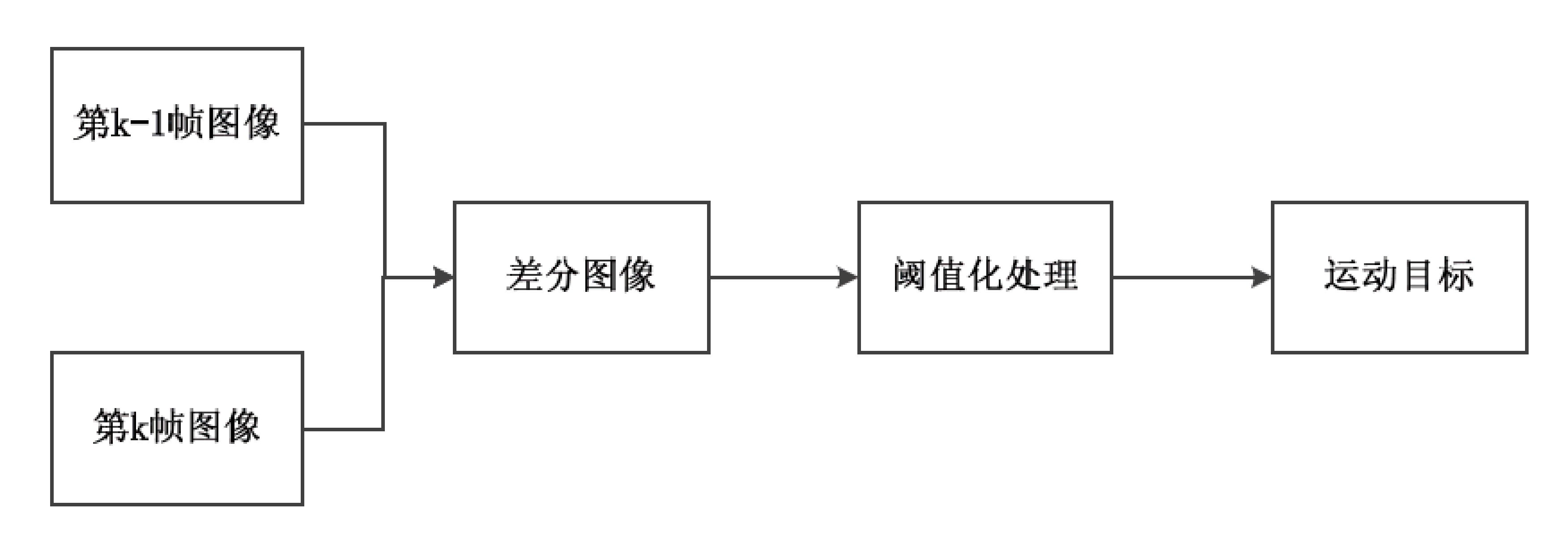


图 11.9　帧差法的基本原理图

选取任意连续两帧图像 $f_k(x, y)$ 和 $f_{k-1}(x, y)$，它们的差分图像 $D_k(x,y)$ 为：

$$D_k(x,y) = |f_k(x,y) - f_{k-1}(x,y)| \quad \text{式（11-1）}$$

其中 (x,y) 表示图像中像素点的空间坐标。将差分图像和一个阈值 T 进行比较，通过比较判定图像中像素点是否为运动目标的像素点，从而提取运动对象 $P_k(x,y)$ 。

$$P_k(x,y) = \begin{cases} 1 \ \ foreground, & D_k(x,y) < T \\ 0 \ \ background, & D_k(x,y) \geqslant T \end{cases} \quad \text{式（11-2）}$$

帧差法原理简单，程序设计复杂度低，易于实现。同时，帧差法能够较强地适应动态环境的

变化，有效地去除由于噪声、光线等因素产生的影响。但是该方法的缺点是无法完全提取出运动目标的所有像素点，无法得到运动目标的完整轮廓，检测结果中运动目标容易出现“空洞”和“重影”现象。因此，检测出的运动目标并不准确，甚至对于运动缓慢的物体无法检测出来。

（3）背景减除法

背景减除法是目前运动目标检测中最常用的一种方法。它的基本思想是：首先通过一定的背景建模方法得到图像的背景模型 $B_k(x,y)$，然后对于视频序列中的每一帧图像 $f_k(x,y)$ 都与背景模型进行差分，对得到的差分图像 $D_k(x,y)$ 进行二值化处理；最后当差分图像中的某个像素点大于阈值T时，就判定该点为运动目标的像素点，否则为背景像素点。用公式表示如下：

$$D_k(x,y)=|f_k(x,y)-B_k(x,y)| \quad \text{式（11-3）}$$

$$P_k(x,y)=\begin{cases}1 \ \ foreground, & D_k(x,y)<T \\ 0 \ \ background, & D_k(x,y)\geqslant T\end{cases} \quad \text{式（11-4）}$$

背景减除法相比于其他方法能够提供最完全的特征数据信息，同时具有复杂度低、实时性好等特点。但在实际应用中，采集到的背景图片随着时间的推移，对外界光线、阴影等因素的影响比较敏感，从而影响目标检测的准确性。因此，基于背景减除法的关键在于寻找理想的背景模型，并对其进行初始化和实时更新，以适应实时环境的变化。

以上为 3 种常用的运动目标检测方法的简单介绍，分别列出了各自的优缺点。本系统采用最常用的背景减除法进行运动目标的检测。

2. 运动目标跟踪

在监控视频领域，运动目标跟踪是后续进行目标识别、行为理解与描述等高级处理的又一关键。运动目标跟踪就是对运动对象进行连续的跟踪以便确定其运动轨迹的过程。目前比较常用的视频运动目标跟踪算法可以分为以下几类：基于 3D 模型的跟踪算法、基于区域匹配的跟踪算法、基于活动轮廓的跟踪算法以及基于特征匹配的跟踪算法。

（1）基于 3D 模型的跟踪算法

该方法的主要思想是先根据经验得到运动目标的三维结构和运动模型，再从实际的图像序列中确定其三维模型参数，从而确定它瞬时的运动参数来实现跟踪。基于 3D 模型跟踪算法的优点在于能够精确地抟系运动目标的三维运动轨迹，但是该方法的运动分析精度受限于几何模型的精度。由于实际应用中很难获得所有运动目标的精确几何模型，因此导致该方法无法得到广泛的应用，并且该方法的计算工作量大，实用性差。

（2）基于区域匹配的跟踪算法

该方法根据图像中运动目标的连通区域的共有特征信息来进行跟踪。该方法的主要思想是将目标图像在当前帧上以不同的偏移值位移，然后根据一定的相似度量准则对每一个偏移值下的重叠的两幅图像、目标图像以及与目标图像同样大小的当前帧图像进行相关处理，根据判别准则与相关处理结果确定目标在实时图像中的位置，即相似度最大所对应的位置就是运动目标位置。

基于区域匹配的跟踪方法实现简单，可以应用于当图像中目标特征不明显且其他方法均失效的场合。但是，一旦连续视频帧图像间光强突变或者噪声较大时，都会对结果造成很大影响。

（3）基于活动轮廓的跟踪算法

该方法的基本思想是将前一帧提取的运动目标辩解轮廓作为轮廓模版，在后续帧的二值边缘图像中跟踪目标轮廓，而且该轮廓可以自动连续地更新。Snake 主动轮廓跟踪算法是近几年发展较快的一种基于轮廓匹配的跟踪算法。该算法利用感兴趣目标轮廓的全局信息，得到一条封闭的轮廓曲线，它被广泛应用于图像分割和分类、边缘提取、运动跟踪等多个领域。但是该算法也存在许多不足，例如初始化轮廓时必须靠近目标的真实边缘，否则将会得到错误的结果，并且该算法对于背景中干扰物与噪声的抗扰性较差，无法对快速运动目标进行跟踪。

（4）基于特征匹配的跟踪算法

基于特征匹配的跟踪算法是监控视频领域较常用的一种跟踪算法。首先从视频序列中选择运动目标的适当特征作为跟踪特征，并且在下一帧图像中根据约束条件来寻找并提取特征，然后将提取到的当前帧中的目标特征与模版特征相比较，根据比较的结果来确定目标。该方法可以忽略运动目标的整体性，仅跟踪一组在运动中具有不变性质的特征点。

海量视频监控检索系统中所用到的 Mean Shift 目标跟踪算法就是一种典型的基于特征匹配的跟踪方法。它将目标颜色灰度等特征信息和 Mean Shift 理论联合起来，对目标形状变化、大小变化等影响不敏感，特征相对稳定，在没有遮挡和部分遮挡的情况下能够准确地进行目标跟踪。该算法具有良好的实用性和稳定性，适用场合广泛，是目前最受关注的跟踪算法之一。

11.3.3　星形骨架方法

在行为识别与规则组合这两个过程都涉及一个重要的方法——星型骨架方法。本系统利用该方法获取行为特征值以及规则特征值，下面详细叙述该方法的处理流程。

1. 星型骨架方法概述

一个人在做不同的动作时，各个动作的姿态存在着巨大的差别，同时，当不同的人在做相同动作的时候，也会存在差别，这些差别正可以区分人体各种运动。人体姿态侧影轮廓的局部极大值点会集在相对于轮廓中心距离最远的区域。通常，人体姿态的侧影轮廓的局部极大值点位于头部和四肢的区域，因此，采用人体姿态侧影轮廓的局部最大值点到侧影轮廓中心的相对距离能够有效地描述人体的姿态，由于这种特征在外形上比较像星状结构，因此，将这种特征命名为星形骨架（star skeleton）特征。图 11.10 描述了星形骨架（star skeleton）特征提取的算法流程。

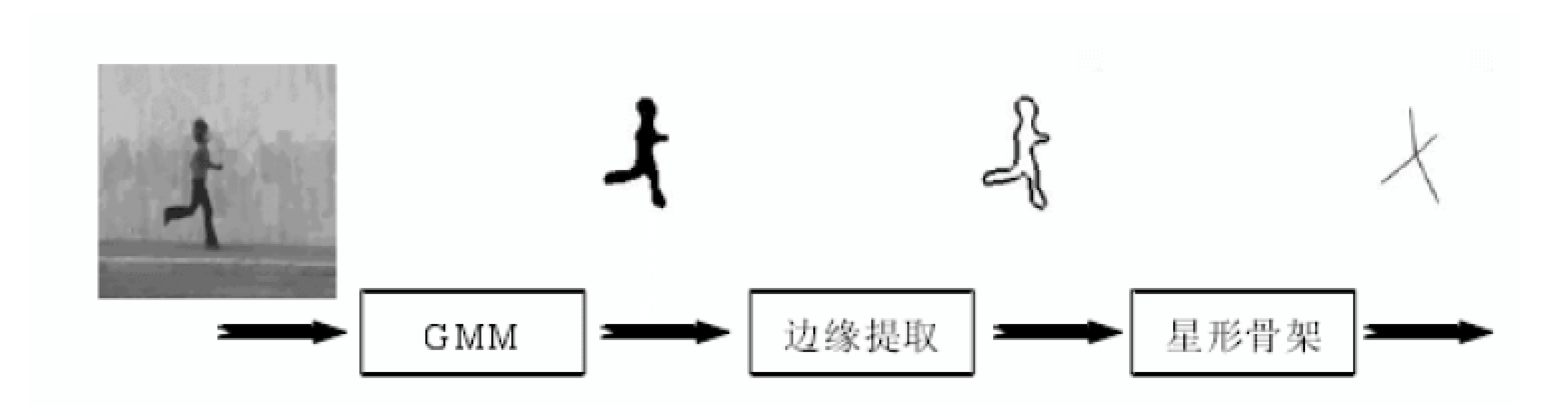


图 11.10　星型骨架（star skeleton）特征提取的算法流程

2. 星型骨架方法的具体实现过程

（1）采用混合高斯模型（GMM）提取运动目标（人）的前景图片

人的运动姿态的特征提取，第一步就是获得人体的前景图像。由于运动姿态的特征提取算法涉及轮廓的提取，所以一般采用混合高斯背景建模（GMM）作为获得前景图像的算法。图 11.11 的原图像来自于以色列 Weizmann 科学院的人体行为数据库中 lena-run1.avi 文件的第 10 帧至第 14 帧。



图 11.11　视频原图像示例

图 11.12 是采用 GMM 提取出的运动目标的前景图像的示例。由于背景更新的关系，一般要到 15 帧左右才能得到比较完整的前景图像。



图 11.12　采用 GMM 提取的前景图象示例

（2）进行前景图象的边缘提取，得到运动目标的轮廓

本系统采用 openCV 的库函数 cvFindContours 来进行轮廓特征的提取，而此函数只能找到大致轮廓点。利用此函数算法对单帧图像中的人体轮廓进行提取，对轮廓边界进行放大，观察发现人体轮廓边界不是单像素点，而是由两三个像素点组成，如图 11.13 所示。因此，若要计算人体质心到轮廓边界的距离必须对轮廓边界单像素化。

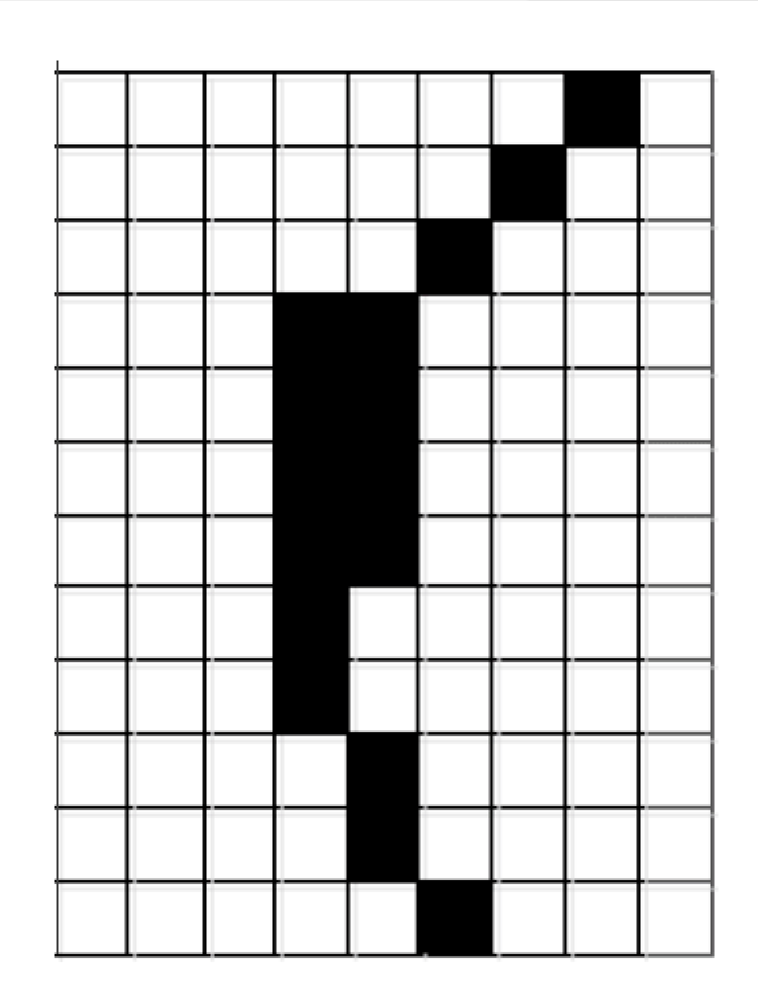


图 11.13　人体轮廓局部方法图

判断一个像素点的下一点可能会有 8 个方向，若该点的相邻点不只一个，则比较所有相邻点与该点的方向和该点与上一个点的方向的夹角，取夹角最小的那个点作为下一个点。

假设图像中水平方向像素点的个数和垂直方向像素点的个数分别为 H 和 V，图像中像素点集合为：$S=\{(x,y)\,|\,0\leqslant x\leqslant H,0\leqslant y\leqslant V\}$。设任意像素点$(x,y)$的灰度值为 g_{xy}，轮廓点集合为 $P=\left\{(x,y)\,|\,(x,y)\in s,g_{xy}\neq 255\right\}$，则非轮廓点集合为 $B=\left\{(x,y)\,|\,(x,y)\in s,g_{xy}=255\right\}$。用一个 3×3 的像素点区域表示候选区域，如图 11.14 所示，对此区域进行标记，则候选区域像素点的集合 $A=\left\{(x,y)\,|\,(x,y)\in U_{i=0}^{8}\{Pi\}\right\}$。

P8 (x8,y8)	P7 (x7,y7)	P6 (x6,y6)
P1 (x1,y1)	P0 (x0,y0)	P5 (x5,y5)
P2 (x2,y2)	P3 (x3,y3)	P4 (x4,y4)

图 11.14　候选区域的标记图

设 P_0 的坐标为(x_0,y_0)，则 P_0 与 $P_i(1\leqslant i\leqslant 8)$的坐标位置关系计算如下：

$$P_1:\begin{cases}x1=x0-1\\y1=y0\end{cases},\ldots\ldots,\quad P_8:\begin{cases}x8=x0-1\\y8=y0-1\end{cases}\qquad 式（11-5）$$

定义以 P_0 为出发点，$P_i(1\leqslant i\leqslant 8)$为终点的向量如下：

$$\overrightarrow{P_{10}}=P_1-P_0,\ldots\ldots,\quad \overrightarrow{P_{80}}=P_8-P_0\qquad 式（11-6）$$

定义向量集合 $Q=\{q\,|\,q=\overrightarrow{P_{io}},1\leqslant i\leqslant 8\}$，设初始点的集合为 $I=\{(x,y)\,|\,(x,y)\in A,y=min\{y_i\},(1\leqslant i\leqslant |A|)\}$，任取 $(xs',ys')\in I$ 为初始点，则初始候选区域像素点的集合为：

$$P_s=\{(x,y)\,|\,(x,y)\in U_{i=0}^{8}\{P_i\},x_0=x_s',y_0=y_s'\} \qquad \text{式（11-7）}$$

设以 (xs,ys) 为候选区域的中心点 P_0，候选区域点集

$P_s'=\{(x,y)\,|\,(x,y)\in U_{i=0}^{8}\{P_i\},x_0=x_s,y_0=y_s\}$，则候选点集合为：

$P_s=A\bigcap P'$，候选向量集合为：$R_s=P\bigcap Q\bigcap P_s$。

设向量 $\overrightarrow{P_{10}}\in R(1\leqslant i\leqslant 8)$，则夹角计算公式如下：

$$a_i'=arccos\frac{\overrightarrow{p_{o0}}^T\cdot\overrightarrow{p_{i0}}}{|\overrightarrow{p_{o0}}^T|\cdot|\overrightarrow{p_{i0}}|},1\leqslant i\leqslant 8 \qquad \text{式（11-8）}$$

其中，a_i' 为向量 $\overrightarrow{p_{o0}}$ 和 $\overrightarrow{p_{i0}}$ 的夹角，为了便于方便，需要对 a_i' 作如下变换：

$$a_i''=\begin{cases}a_i' & if\ a'\geqslant 0\\ \pi+a_i' & otherwise\end{cases}$$

向量夹角的集合为 $D=\{b\,|\,b=a_i'',1<i<8,b\neq\pi\}$

设提取的轮廓点集合为 E，则 $E=E\bigcup(x_s',y_s')$，取 $a=\min D(1\leqslant i\leqslant 8)$。按照上述流程即可得到轮廓的单像素化。最后得到运动对象的单像素化的轮廓如图 11.15 所示。

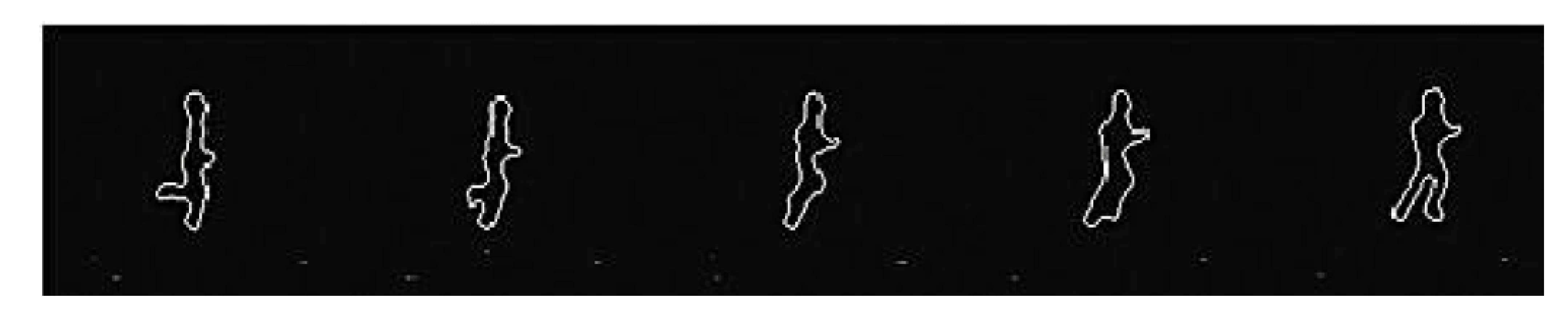


图 11.15　图像序列的轮廓示例

（3）采用星型骨架算法，提取运动目标的星型骨架

得到了运动目标姿态的完整轮廓，就可以采用星形骨架算法得到运动物体的星形骨架。算法流程图如 11.16 所示。

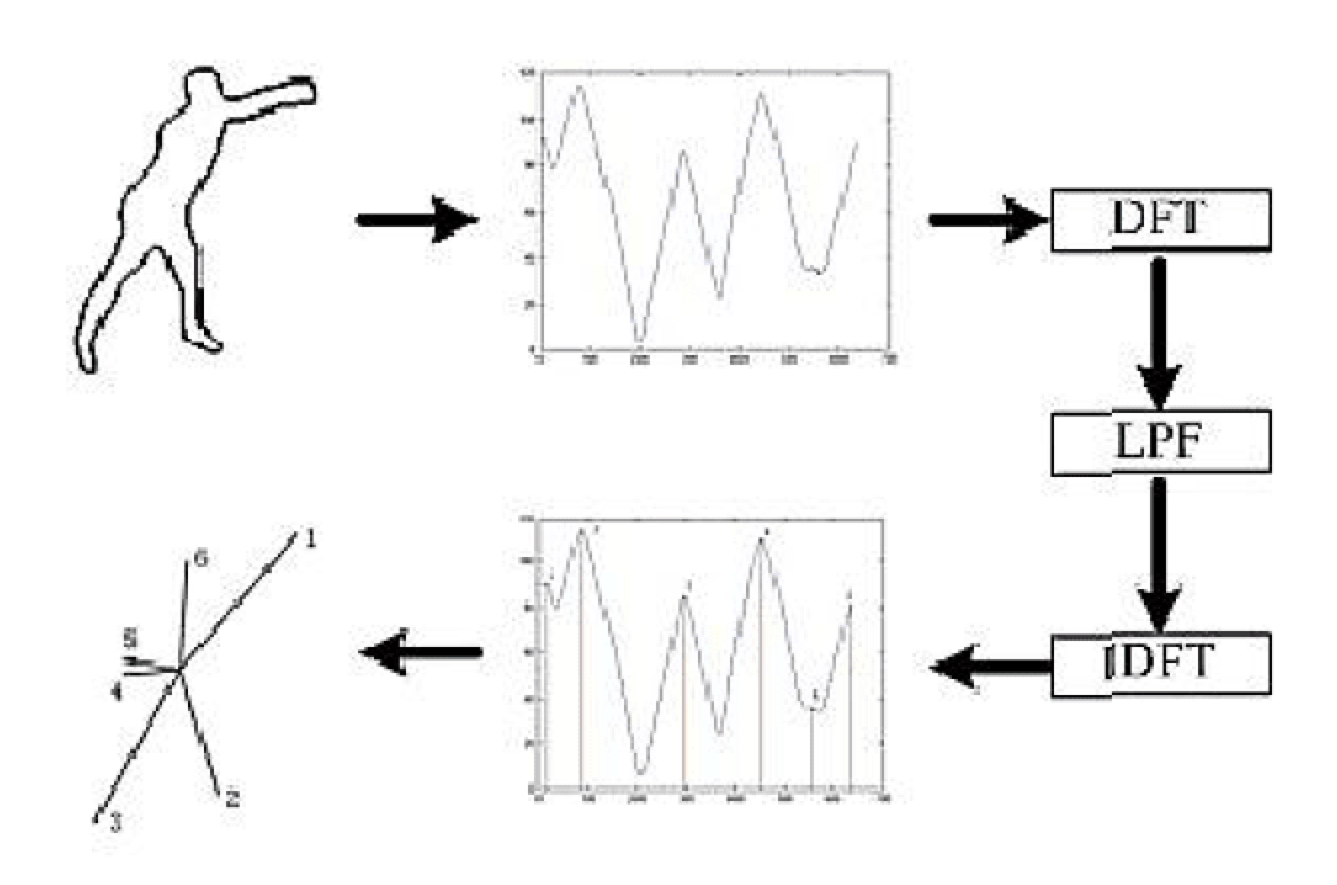


图 11.16　星型骨架算法流程图

该算法主要由以下 4 部分组成。

- 基于获得的姿态轮廓得到轮廓的质心。轮廓的质心获得根据以下公式得到:

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \text{式（11-10）}$$

$$y_c = \frac{1}{N}\sum_{i=1}^{N} y_i \qquad \text{式（11-11）}$$

其中，N 为轮廓边界点的数目，(x_i, y_i) 为轮廓边界点的坐标，(x_c, y_c) 为获得的轮廓质心。

- 计算轮廓上的点到轮廓质心的欧式距离 d_i。利用 x 和 y 计算轮廓上的每一个点与轮廓质心的欧式距离，得到一个一维的数字序列 d_i。

$$d_i = \sqrt{(x_i - x_c) + (y_i - y_c)^2} \qquad \text{式（11-12）}$$

- 对 d_i 进行平滑处理。采用平滑滤波器或低通滤波器对 d_i 进行平滑处理，以减少噪声的干扰，得到平滑后的输出序列^d_i。
- 寻找出^d_i 的局部极大值点与轮廓质心获得运动目标的星形骨架。根据^d_i>^d_i+1 且^d_i>^d_i−1 得到局部最大值点，获得运动目标的星形骨架。

图 11.17 是 lena-run1.avi 视频中第 10~14 帧提取的星形骨架。选用星形骨架算法作为提取人的运动姿态特征主要有两个优势：第一，有效地减少了特征数据的冗余度，减少了计算量。第二，通过选择合适的低通滤波器或者平滑滤波器，可以有效地降低特征向量的维数。在本系统中，采用的是指数滤波器，通过调节系数，将特征向量的维数降低到 5 维左右，这是因为每个人所做的各种姿态，手脚之间的摆动幅度是不一样的，可以通过这种差别，有效地描述人的运动姿态。图 11.17 是采用星形骨架算法从上面的图中对应运动序列的星形骨架特征。

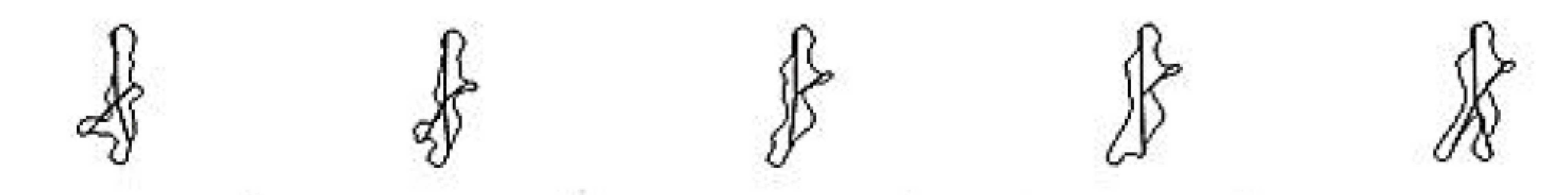


图 11.17　获取的星型骨架特征

通常，通过分析人体骨骼各部分的运动来确定运动目标的状态目前主要应用在人体行为分析或者是步态研究。通过星形骨架的提取，得到了人体运动姿态的表征，但这只是第一步，下面就是怎样将这种表征转化为可以度量的特征向量形式。由于根据星形骨架特征，得到了人体前景轮廓的局部最大值点分别代表人体的头、手、脚所在的区域，采用两个脚之间的夹角被用来区分人的姿态是走还是跑，但是无法区分更多的人体运动姿态。

由于人存在高矮胖瘦，因此，在选择特征向量时，应该是特征向量具有普遍适应性，因此，可以通过计算提取出来前景图像轮廓的局部最大值点与轮廓质心的幅角作为特征向量的元素。之所以选择前景图像轮廓的局部最大值点与轮廓质心的幅角作为特征，是因为它抛开了人的姿态由于高矮胖瘦的影响，可以较好地定义一个人的运动姿态。如图 11.18 所示，对于提取到一帧图像的星形骨架对应的运动特征向量为 A={48.667789，287.311646，240.788712，184.899094，170.272415，85.732109}，其中特征向量 A 中的各个元素分别为各个局部最大值点与轮廓质心的幅角。

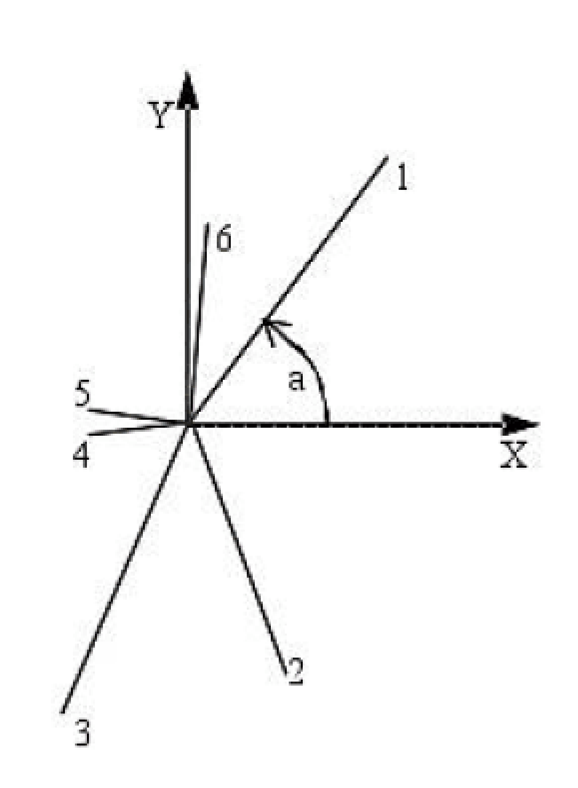


图 11.18　星型骨架特征的参数化

以上介绍为一般星型骨架方法的主要处理流程，也是本系统所采用的主要行为特征提取方法。有所变动的是，本系统在提取出单像素化轮廓并计算出质心与欧式距离后，按照人体的头部以及上下肢之间的比例，将人体分为 5 部分。如图 11.19 所示，分为 A、B、C、D、E 共 5 部分。对每一部分计算欧式距离的最大极大值点，然后建立如图 11.20 所示的以质心为原点的坐标系，计算出 5 个最大极大值点在该坐标系中的角度偏移量。假定 5 个最大极大值点的坐标 $(x_i, y_i)(1 \leqslant i \leqslant 5)$，质心坐标为 (x_c, y_c)，则计算出最大极大值点与质心坐标的角度 $\theta_i(1 \leqslant i \leqslant 5)$偏移量公式如下。

$$\theta_i = \frac{a\tan\left(\frac{|y_i - y_c|}{|x_i - x_c|}\right)}{3.14} \times 180 \quad \text{式（11-13）}$$

最后以 5 个最大极大值点的角度偏移以及质心坐标，组成一帧图片中一个对象的行为特征值。

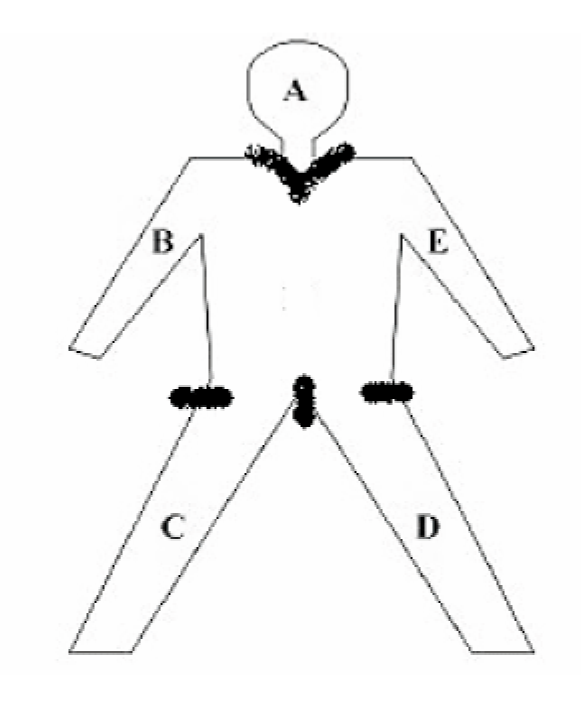


图 11.19　人体躯干划分

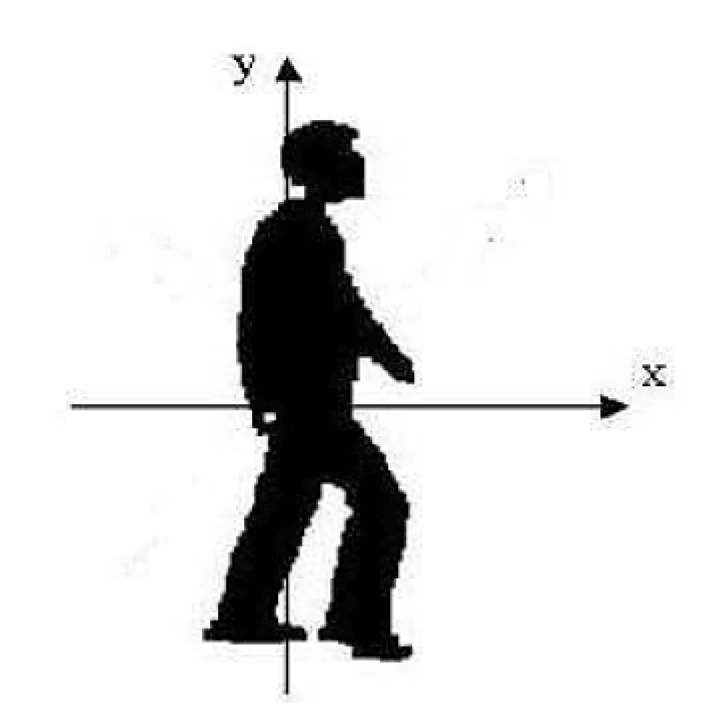


图 11.20　建立以质心为原点的人体坐标系

11.4 详细设计与实现

11.4.1　基于 MapReduce 的视频预处理

1. 利用 GMM（混合高斯模型）进行背景建模

在 11.3.2 节中已经对从视频中检测运动目标的方法做了简单的介绍，并指出本系统采用背景减除法中的 GMM 算法进行运动目标检测。接下来将详细阐述 GMM 背景建模与目标检测的实现。

混合高斯背景建模的基本思想是：将图像中的每一个像素点都由 K 个高斯分布来表示。其中 K 的取值一般为 3~7，这是由计算机内存和对算法速度的要求来决定的，K 值越大则处理波动能力就越强但是处理时间也会越长。令某一像素点在 t 时刻的颜色取值表示为 X_t（灰度图像中 X_t 是一个标量，彩色图像中 X_t 是一个矢量），其概率密度函数用 K 个高斯函数的线性组合表示：

$$P(X_t) = \sum_{i=1}^{k} \omega_{i,t} \times \eta(X_t, u_{i,t}, \sum_{i,t}) \quad \text{式（11-14）}$$

$$\eta(X_t, u_{i,t}, \sum_{i,t}) = \frac{1}{(2\pi)^{n/2} |\sum_{i,t}|^{n/2}} e^{-\frac{1}{2}(X_t - \mu_{i,t})^T \sum (X_t - \mu_{i,t})} \quad i = 1, 2, \cdots, K \quad \text{式（11-15）}$$

其中，$\eta(X_t, u_{i,t}, \sum_{i,t})$ 是第 i 个高斯分布，$\omega_{i,t}$ 表示时刻 t 混合高斯模型中第i个高斯分布的权系数的估计值，其大小体现了当 i 按模型表示像素值时的可靠程度，同时满足 $\sum_{i=1}^{k} \omega_{i,t} = 1$；$u_{i,t}$ 和 $\sum_{i,t}$ 分别表示时刻 t 混合高斯模型中第 i 个高斯分布的均值向量和协方差矩阵，均值向量体现了每个

高斯分布的中心，协方差是高斯分布的宽度大小，体现了像素值的不稳定程度；n 是 X_t 的维数，当为灰度图像时，n=1，当为彩色图像时，为了提高算法的效率并降低复杂度，假定每帧图像的各个像素点的 RGB 三个颜色分量均是相互独立的，且具有一样的方差，其协方差矩阵的取值为：

$$\sum_{i,t}=\begin{bmatrix}\sigma_r^2 & 0 & 0\\ 0 & \sigma_g^2 & 0\\ 0 & 0 & \sigma_b^2\end{bmatrix}\qquad i=1,2,\cdots,K \tag{式（11-16）}$$

式中 σ_r^2、σ_g^2、σ_b^2 分别为对应颜色分量的高斯模型方差，即表明每一个颜色通道各建立了一个一维的混合高斯模型。

高斯混合模型在使用之前必须进行初始化，为了降低对计算机性能的要求，直接将每个高斯分布的权系数和均值向量都初始化为 0，协方差赋予一个初始值 VO。接下来开始根据时间序列上像素的动态变化来更新混合高斯背景模型，此时需要将新图像的像素值 X_t 与 K 个高斯分布分别通过以下依据进行匹配：

$$|X_t-u_{i,t-1}|\leqslant 2.5\sigma_{i,t-1} \tag{式（11-17）}$$

如果像素值 X_t 与某个高斯分布的均值 $\mu_{i,t-1}$ 之间的差值在相应分布标准差的 2.5 倍范围之内，则认为该像素值与该高斯分布匹配，否则不匹配。与当前像素 X_t 匹配的高斯模型时需要对 K 个高斯模型进行动态更新，其更新公式如下：

$$\omega_{i,t}=(1-\alpha)\omega_{i,t-1}+\alpha \tag{式（11-18）}$$

$$\mu_{i,t}=(1-\beta)\mu_{i,t-1}+\beta X_t \tag{式（11-19）}$$

$$\sigma_{i,t}=(1-\beta)\sigma_{i,t-1}^2+\beta(X_t-\mu_{i,t})^T(X_t-\mu_{i,t}) \tag{式（11-20）}$$

其中，$\alpha(0<\alpha<1)$ 为自定义的学习速率，其大小决定了背景的更新速度，本系统中 $\alpha=0.3$；β 为参考学习速率，表示高斯分布中各参数更新的快慢，β 可以近似等于 $\alpha/\omega_{i,t}$。

如果像素值 X_t 没有与任何高斯分布匹配，则引入一个新的高斯分布替换其中对应权值最小的高斯分布，该高斯分布以当前像素为均值，重新初始化一个较大的方差和一个较小的权值，而其他高斯分布 $\mu_{i,t}$、$\sigma_{i,t}$ 参数保持不变，只对其权值进行以下处理：

$$\omega_{i,t}=(1-\alpha)\omega_{i,t-1}\qquad \alpha\text{ 为学习率} \tag{式（11-21）}$$

混合高斯模型只对图像中的所有像素进行建模，并不对背景与前景进行判别。但是，根据参数学习机制可知，由于背景像素值出现率较高，因此由那些权值大的高斯公式来表述，而运动目标采用权重较小的高斯分布进行表述。在获得新的视频帧后，首先需要根据新图像的像素值对混

合高斯模型进行更新，然后归一化所有高斯函数的权重 $\omega_{i,t}$，并将 K 个高斯模型根据 $\frac{\omega_{i,t}}{\sigma_{i,t}}$（此值越大则说明像素出现的概率越大，是背景的可能性越大）进行降序排列。设置阈值 T，$\frac{\omega_{i,t}}{\sigma_{i,t}}$大于 T 的分布则均为背景分布，其他则为运动前景分布。

混合高斯背景建模尤其适合室外光线以及天气变化不太强烈的情况下的运动目标检测，而且能够很好地判断动静相互转换的物体，具有很强的自适应能力。本系统采用 OpenCV 函数库中提供的混合高斯模型进行运动目标提取。由于 MapReduce 编程框架对于 Java 有更好的支持，因为本系统中后续将使用利用 Java 封装后的 OpenCV 库进行视频的处理，但是为了方便描述，此处直接给出 C/C++版本的函数。

BackgroundSubtractorMOG2 为函数库定义的混合高斯模型类，通过该类就可以实现运动前景与背景的分离，主要的步骤如下：

```
BackgroundSubtractorMOG2 bg_model;  //定义一个混合高斯模型类
for(;;)
{
//img 与 fgmask 分别为当前帧与前景图片，根据此构造函数即可进行前景与背景分离
 bg_model(img, fgmask, update_bg_model ? -1 : 0);
//可以通过 getBackgroundImage 方法得到背景图片
bg_model.getBackgroundImage(bgimg);
}
```

2. 运动目标检测的 MapReduce 设计

海量监控视频检索系统分为由视频文件触发的视频处理模块与由用户触发的视频检索模块。由视频文件触发的视频处理模块在不需要人工干预的情况下，只要监控摄像头在不停地产生视频文件，该模块就会持续运行下去，对每个视频文件中的运动对象进行检测与提取。本系统基于“海量”的监控视频，即系统需要处理的是不止一个监控摄像头产生的视频文件，以一个监控摄像头 10 分钟产生一个监控视频为例，一个监控摄像头一天将会产生 144 个视频文件，10 个摄像头一天将会产生 1440 个视频文件。因此，如果用户在运用本系统之前就已经安装了监控设备，则会存在海量的没有经过处理的原始视频数据。

如果采用串行的方式进行这些原始视频文件的处理，对于海量原始视频数据的处理所需的时间将会是相当长的，因此，本系统选择使用 MapReduce 并行分布式处理框架对海量的视频数据进行处理，以提高处理的效率，如图 11.21 所示为视频预处理模块的体系结构。适合用 MapReduce 来处理的任务有一个基本要求：待处理的数据集可以分解成为许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。

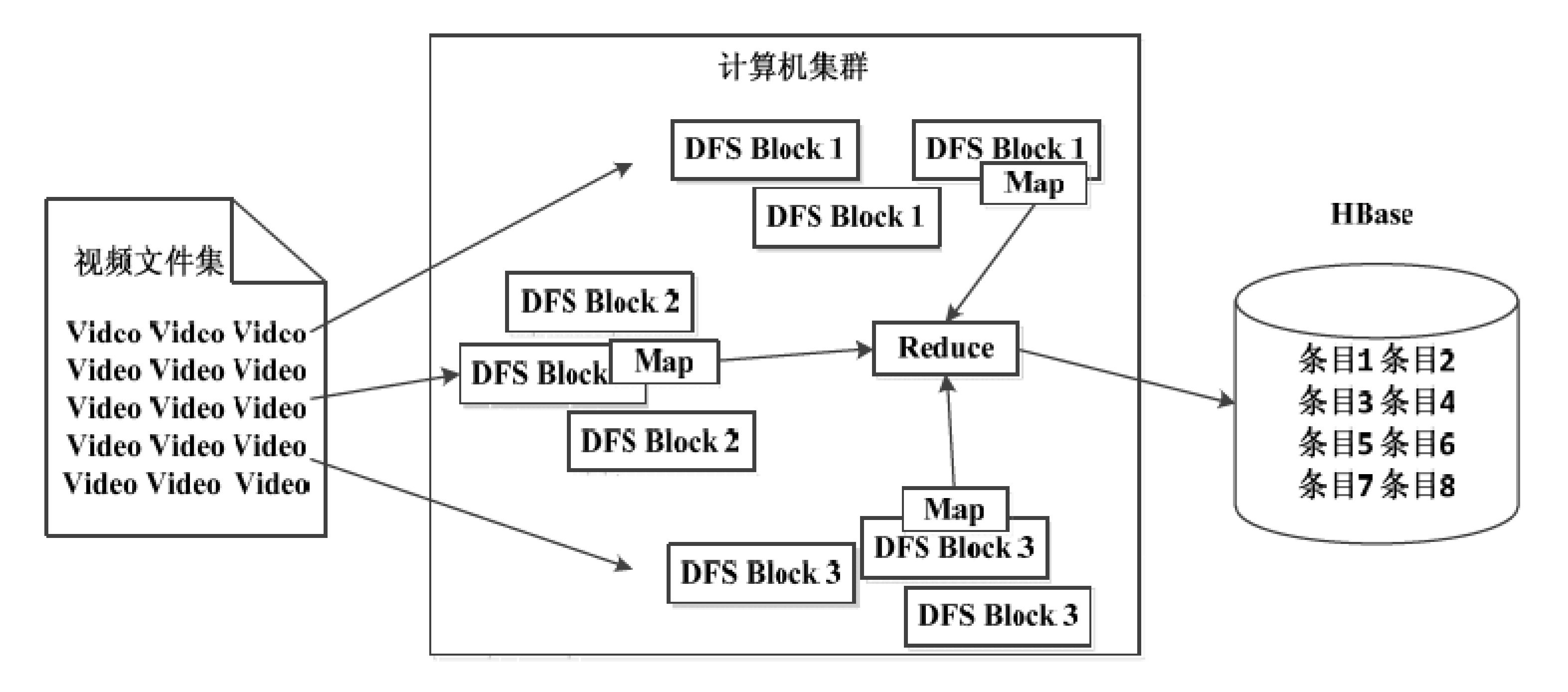


图 11.21　视频预处理模块体系结构

大多数 MapReduce 程序的编写都可以简单地依赖于一个模板及其变种。使用这个模板编写 MapReduce 程序，主要需要实现两个函数：继承自 Mapper 的 map 函数以及继承自 Reducer 的 reduce 函数。一般遵循以下格式：

- Map:　(k1,v1) –> list(k2,v2)

```
public static class Map extends Mapper<K1, V1, K2, V2> {
public void map(K1 key, V1 value, Context context)
throws IOException { }
}
```

- Reduce:　(k2,list(v2)) ->list(v3)

```
public static class Reduce extends   Reducer<K2, V2, K3, V3> {
public void reduce(K2 key, Iterator<V2> values, Context context)
 throws IOException { }
}
```

针对本系统来说，待处理的视频集可以分解成许多的单一视频，而且每一个视频文件都可以完全并行地进行处理。接下来将针对如何将视频处理模块全部或者部分转换到 MapReduce 编程模型中的两个计算模型中进行设计与实现。

视频预处理的目标是要对指定文件夹中的视频文件进行处理，提取出每个视频中的运动目标（object_id）、视频生成时间（time）、起始帧（start）、终止帧（end）、目标图片（pic）等运动对象与视频的属性，然后将这些属性均存储到分布式数据库 HBase 中。图 11.22 画出了利用 MapReduce 模型进行视频预处理的流程图。

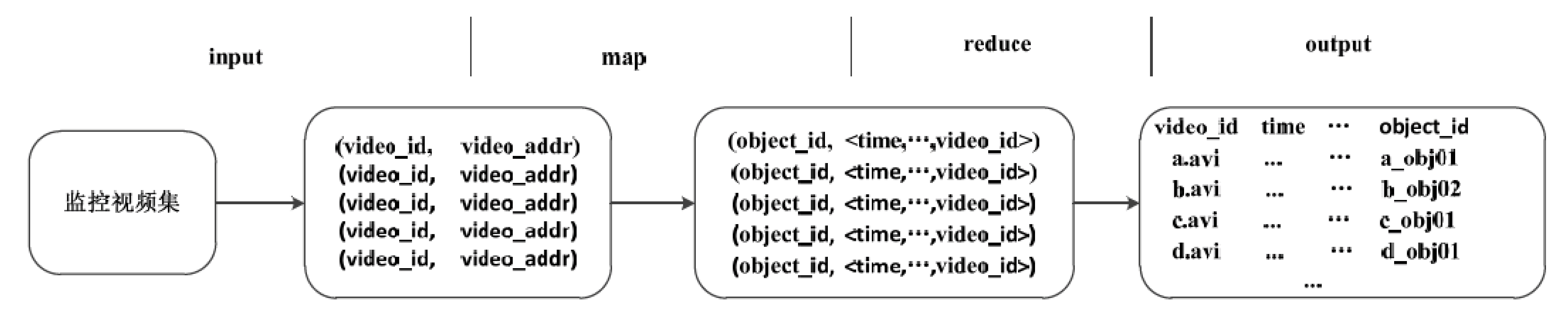


图 11.22　利用 MapReduce 模型进行视频预处理流程图

首先，通过监控摄像头产生视频文件，并保存在本地物理磁盘上的同时通过建立索引将视频文件按摄像头编号存储到 HDFS 文件系统中。以 10 分钟的监控视频文件为例，其单个大小约为 50MB，而 MapReduce 首先将输入文件划分为 M 个 16MB~64MB 的数据分片，则说明单一视频可以被作为一个数据分片进行处理。其次，为自定义的 map 函数制定输入文件的内容格式，这里需要自定义一个 InputFormat 类。由于 map 函数的输入必须是 key/value 对，本系统自定义的 map 函数指定输入 key 为视频文件名 video_id，是 Text 类型，value 为所处理的视频文件的物理地址信息，也是 Text 类型。由于默认提供的操作无法满足我们的需求，所以我们继承 InputFormat 类，将输入的文件集解析为 map 函数所需要的 key/value 值形式。

然后，自定义 map 函数承担 map 任务的实现。在该函数中接收刚刚解析出的 key/value 对，再提取 value 值中保存的视频文件的物理地址，对视频文件进行运动目标以及属性提取的处理，并组合成中间 key/value 值传递给 reduce 函数进行处理。其中，key 为每个运动对象的唯一 id，value 值是一个关联数组，它使用 Hadoop 特有的关联数组 MapWritable，其中保存 video_id、time 等内容。输出类型为<Text,MapWritable>。

最后，需要自定义 reduce 函数实现 reduce 任务。此处 reduce 任务相对简单，只是将 map 函数产生的中间 key/value 值存储到 HBase 中。经过以上操作，HDFS 中的监控视频文件集先由自定义的 InputFormat 分割为单个的视频文件，再经过 map 操作得到每个视频文件中的 object_id、time、object_pic 等信息，最后由 reduce 操作将每个视频文件的信息存储到 HBase 中，这样就利用 MapReduce 框架完成了视频预处理的逻辑过程。由于 map 任务和 reduce 任务都是分布式的高度并行的，所以 MapReduce 框架在处理海量监控视频检索系统的大量数据上有着不可忽视的优势。

3. MapReduce 对运动目标检测的实现

根据上述的详细设计流程，海量监控视频检索系统的视频预处理功能需要以下 2 个核心类帮助实现。

（1）VideoMapper：自定义的 Mapper 类。该类继承 Mapper 类，并重写 Mapper 类中的 map 函数，主要实现对原始视频数据的 map 操作。在这里是对参数中传递的 value 值所对应的视频数据进行处理，分析出视频数据中的运动目标、视频生成时间、目标图片等信息。

（2）VideoReducer：自定义的 Reducer 类。该类继承了 Reducer 类，并重写 Reducer 类中的 reduce 函数，主要完成对中间数据集的合并任务。本例中 reduce 函数功能较为简单，所需完成的只是将 map 函数产生的中间 key/value 值存储到 HBase 中。

VideoMapper 和 VideoReducer 是以 MapReduce 编程模型实现视频预处理功能中的最重要的两个类，分别实现了 MapReduce 编程模型中的 map 操作和 reduce 操作，如图 11.23 是系统视频预处理模块的类交互图。

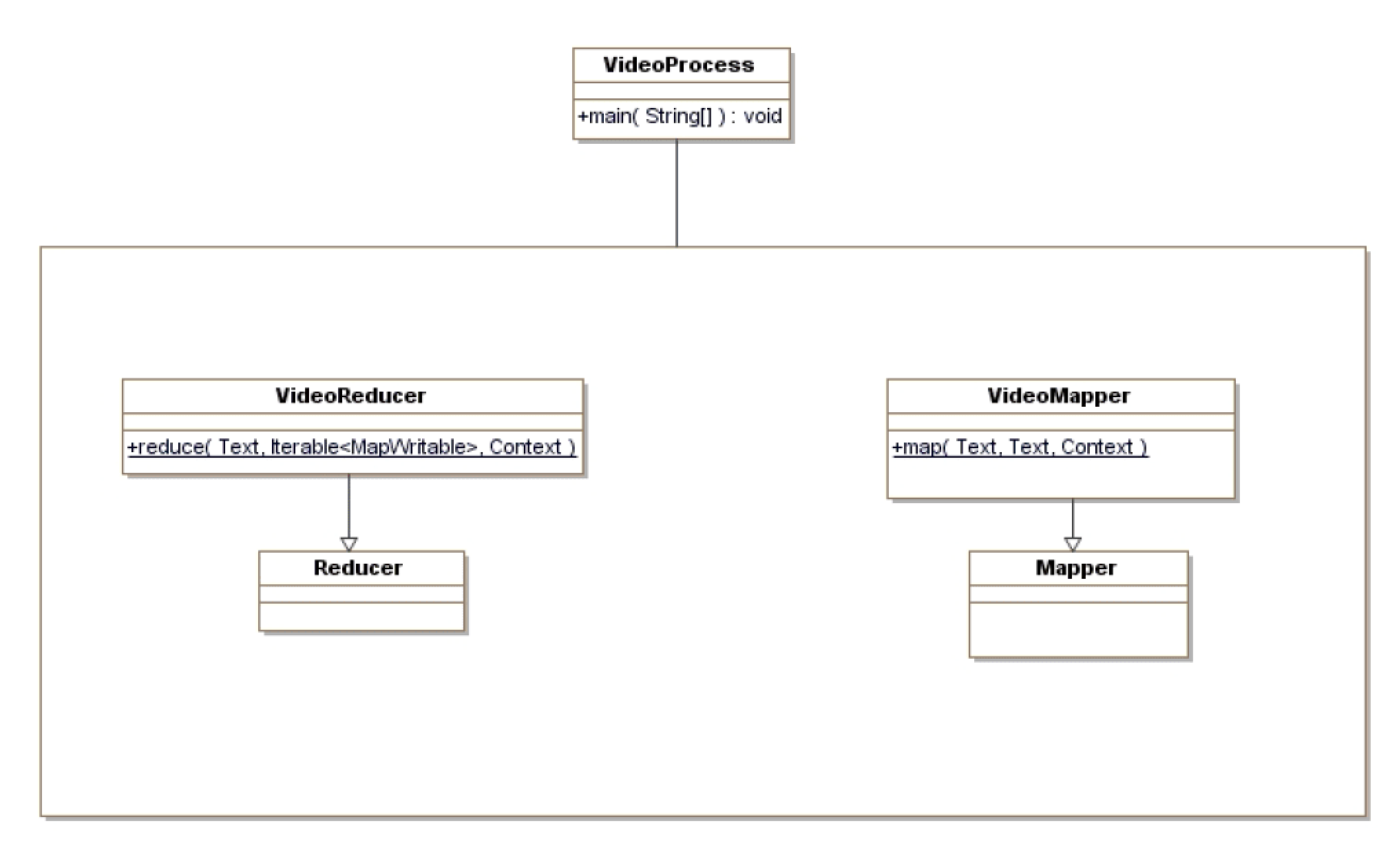


图 11.23　系统视频预处理模块的类交互图

由于 Hadoop 提供了强大的接口才使得程序员可以十分简便地实现 MapReduce，程序员只需要表述他想要执行的简单 map 和 reduce 运算即可，不必关心并行计算、容错、数据分布、负载均衡等复杂的细节问题。

在读取了视频数据后，需要解析 key/value 值作为 Mapper 的输入，进入到 map 阶段。VideoMapper 类实现了 map 操作，在这个操作里，首先根据输入的 value 值对视频文件进行处理，这里可以直接调用已经封装好的用于运动目标检测的动态链接库文件，得到视频中存在的运动目标、视频生成的时间等信息。执行完 map 过程后，所有具有相同 key 值的中间 value 随后会将 Hadoop MapReduce 框架进行组合，并被传输到自定义的 Reducer 类中。VideoMapper 类利用分而治之的思想，将整个文件集分割成为单个的视频进行并行处理。在每个 map 操作中，利用 GMM 建模与 Mean Shift 算法得到运动目标、运动轨迹以及目标图片等信息，并利用 OpenCV 提供的库函数得到所处理的视频的生成时间、视频帧率等视频信息，将视频与运动目标这两部分的信息通过自定义的 Reducer 类存储到 HBase 中。

在 main 函数中用户需要定制一个 Job，用于执行一次计算任务，可以通过一个 Job 对象设置如何运行这个 Job。下面代码中，定义了输出的 key 类型是 Text，value 的类型是 MapWritable，指定自定义的 VideoMapper 类作为 Mapper 类，使用自定义的 VideoReducer 类作为 Reducer 类。任务的输入路径和输出路径由命令行参数指定，这样计算任务运行时会处理输入路径下的所有文件，并将计算结果写到输出路径中。最后通过 Job 对象的 waitForCompletion（）方法运行计算任务，并一直等待直到计算结束，以此来完成一次 MapReduce 计算。

```
public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException
{
```

```
Configuration conf = new Configuration();
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
if (otherArgs.length != 2) {
     System.err.println("Usage: VideoProcess <in><out>");
     System.exit(2);
 }
Job job = new Job(conf, video process");
job.setJarByClass(videoprocess.class);
job.setMapperClass(VideoMapper.class);
job.setReducerClass(VideoReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(MapWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
job.waitForCompletion(true);
}
```

11.4.2　基于 HBase 的视频数据存储

1. MEPG-7 与基于内容的海量视频检索系统

在 11.3.1 节中对 MPEG-7 以及 OpenCV 图像处理库进行了简单介绍。在本节中，我们将对这两者在系统中的具体应用进行详细的阐述。

视频检索就是从大量的视频数据中找到所需的视频片断或视频点。传统的视频检索只能通过快进和快退等方法人工查找，是极为繁琐且耗时的一项工作，这显然已无法满足多媒体数据库的要求。用户往往希望只要给出示例图像或给出特征描述，系统就能自动地找到所需的视频片断点，即实现基于内容的视频检索。

基于内容的视频检索是建立在基于内容的静态图像检索基础上的，因为视频可以看作是一个连续静态图像的序列，其中的每一幅静态图像称为一帧。在基于内容的海量视频检索系统中，由于针对的是安防监控领域的应用，从监控摄像头中获取的原始视频有如下特点：背景图片基本上不变或者是变化很小的，我们只关注镜头中运动的对象。所以在本系统中，只提取识别出有运动物体的图像帧。系统采用 OpenCV 图像处理库提取对象图片以及颜色特征、纹理特征以及运动特征。如果读者感兴趣的话，利用 OpenCV 进行操作的具体过程可以参考《学习 OpenCV》这本书，这里不做详细介绍。

接下来，向读者介绍 MPEG-7 标准的视频描述工具，包括用来描述视频对象的各种描述符和描述方案。它的组成部分包括网格结构、2D-3D 多视角、时间序列、空间 2D 配合、时间插补这 5 种基本结构和几种基本的视觉特征描述符：颜色、纹理、形状、运动、定位及其他。每一类都由基本和复杂的描述符和描述方案组成。由于基于内容的海量视频检索系统主要涉及颜色描述符、

纹理描述符以及形状描述符，因此重点介绍这三种描述符。

（1）颜色描述符

MPEG-7 定义的颜色特征包括 7 种颜色描述符，分别是颜色空间（Color Space）、颜色量化（Color Quantization）、主颜色（Dominant Colors）、可伸缩颜色（Scalable Color）、颜色布局（Color Layout）、颜色结构（Color Structure）和帧组 / 图像组颜色（GoF / GoP Color）。其中颜色空间和颜色量化这两个描述符主要用来和别的描述符相结合，不单独使用，它们规定了别的描述符使用的颜色空间和量化的方法。

- 颜色空间：MPEG-7 标准颜色描述符定义的颜色空间，支持的几种颜色空间模型有 RGB、YCbC、HSV、HMMD、单色以及对 RGB 的线性转换阵模型等。
- 颜色量化：该描述符与主颜色等其他特征描述符配合使用定义颜色空间的均匀量化方法。它包括线性量化、非线性量化以及查找表量化。量化时可以根据不同应用规定量化比特数。
- 主颜色：该描述符最适用于表示局部（对象或图像区域）特征，它也可以用于整个图像，例如旗帜图像或彩色商标图像. 可以通过颜色量化等方法提取每一区域或图像的少数几种出现频率较高的表示色作为主颜色，来进行图像颜色信息的相似性检索。
- 可伸缩颜色：可伸缩颜色是 HSV 空间的颜色直方图，定义了一个可调的二进制的位数和可调的表达精度来表示图像的颜色特征. 这个描述符主要用于图像与图像的匹配和基于颜色特征的检索，检索的精度随着描述中使用的比特数目的增加而增加。
- 颜色布局：指定了图像颜色的空间分布模型，以帮助提高检索和浏览的速度。既可以定义整幅图像的颜色空间分布，也可用于一幅图像的某一部分的颜色分布。
- 颜色结构：颜色结构描述符是一个颜色特征描述符，它既包括颜色内容信息（类似于颜色直方图），又包括内容的结构信息。它的主要功能是图像与图像的匹配，主要用于静态图像检索。
- 帧组 / 图像组颜色：该描述符将静态图像的可伸缩颜色描述符扩展到对视频片断或静态图像集合的颜色描述。所定义的帧组可以是镜头、子镜头或者是镜头组。

（2）纹理描述符

MPEG-7 中定义了三种纹理描述符：同构纹理描述符（Homogenous Texture Descriptors）、纹理浏览描述符（Texture Browsing Descriptors）和边缘直方图描述符（Edge Histogram Descriptors）。

- 同构纹理：同构纹理作为一个重要的视觉基本特征，反映了纹理的一致性和空间频度分布，主要用于大量相似图案的搜索和浏览。一幅图像可看作由规则纹理以马赛克形式拼接而成，所以与这些区域关联的纹理特征可以作为索引来检索图像。
- 纹理浏览：纹理浏览描述符对于表现纹理特征的类型浏览很有用，并且它最多仅需 12 比特信息。类似于人的特征区分，这个描述符从结构性、分布粒度和方向性等方面表现纹理的感知特征。
- 边缘直方图：边缘直方图描述符表示 5 种类型边缘的空间分布图——纵向、横向、45 度、135 度以及非方向。它可以用于相似语义的图像检索，因此它的主要目标在于图像与图像

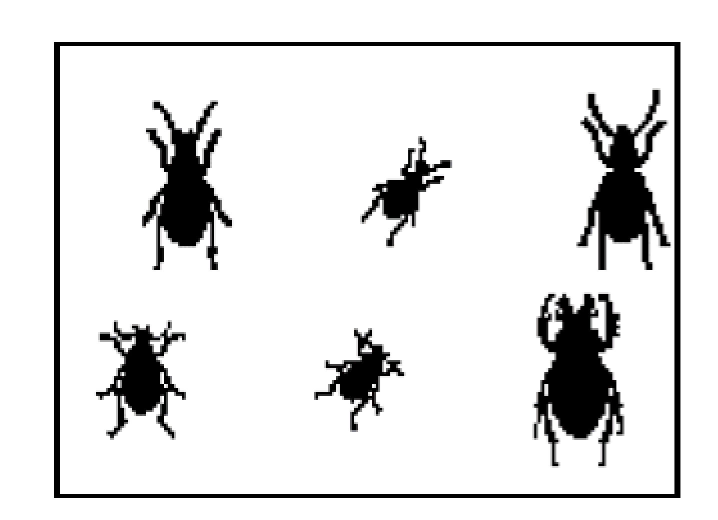
的匹配（通过示例或草图），特别是边缘分布不规则的自然图像。

（3）形状描述符

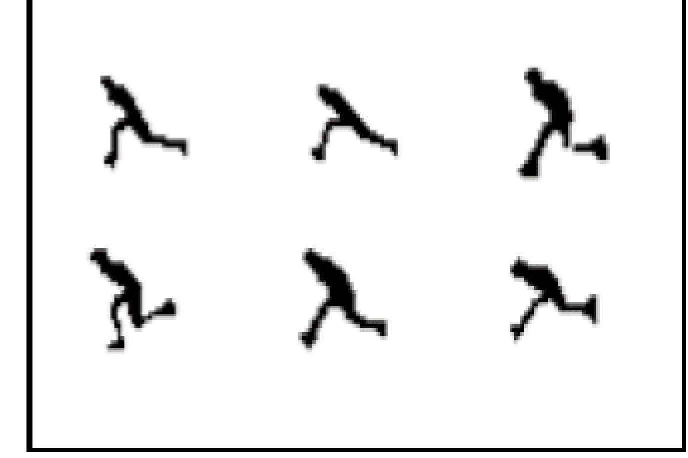
形状特征对于人来说是识别物体的主要信息，是一种重要的图像内容表达手段。不同于颜色或纹理等底层特征，形状特征的表达必须以对图像中物体或区域的划分为基础。MPEG-7 定义了三种形状描述符：基于区域的形状（Region Shape）、基于轮廓的形状（Contour Shape）和三维形状（Shape 3D）。

- 基于区域的形状：可用于描述单个连通的区域或多个不连通的区域，由一组角放射变换系数（Angular Radial Transformation，ART）构成。基于区域的形状描述符具有小的数据量、快速的提取时间和匹配等特点，表示该描述符的数据大小固定为 17.5 字节。显然，特征提取和匹配过程能够拥有较低的计算复杂度，因此适合于在视频数据处理中跟踪外形。
- 基于轮廓的形状：基于轮廓的形状描述所描述的对象或区域的形状特征是以它的轮廓线为基础，它常常被称为曲率尺度空间（CSS）表示，重在描述对象的具有感知意义的形状特征。它包括一些重要的特点，即：
  - 在基于相似性的检索中，它能够捕捉到很好的形状特征。它能够捕捉到很好的形状特征。它能够捕捉到很好的形状特征。它能够捕捉到很好的形状特征。
  - 它反映了人类视觉系统的感知性能，并提供良好的泛化。
  - 它非常适合描述非刚体运动。
  - 能较好地支持有局部遮挡的形状。
  - 很好地支持在视频和图像中常出现的因为相机参数的变化而形成的角度转换。
  - 它的表述非常紧凑。

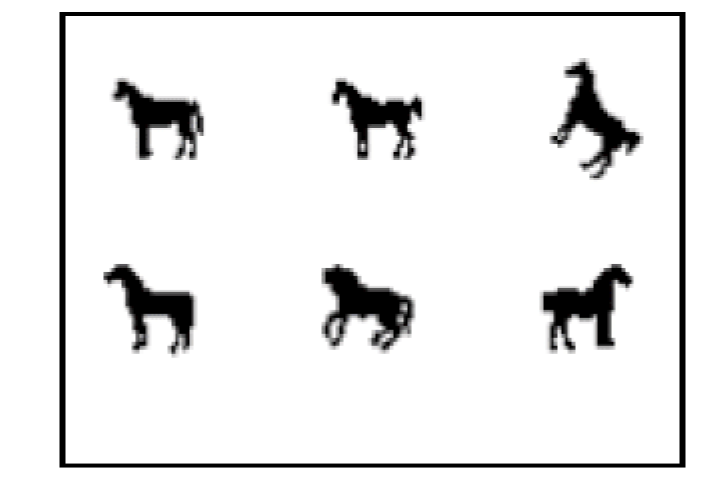
这个描述符中的上述某些性能在图 11.24 的几组图片中得到体现，从 MPEG-7 的形状数据库中，根据实际的检索结果发现图片每帧含有非常相似的 CSS。



(a) 形态综合性能



(b) 鲁棒性的非刚性运动



(c)部分遮挡的鲁棒性

图 11.24　描述符的性能展示

- 三维形状：提供了 3D 网络的本征形状描述，基于三维形状索引（表达三维曲面的局部曲率特性）来表示图像形状特征的直方图。主要用于在 3D 模型数据库中进行搜索、检索和浏览。接下来，向读者介绍 MPEG-7 描述文档的组成，它通常包括三部分：头部声明、模式（可选，用来定义文档中的标记及语法关系等）以及文档实例。

```
<<?xml version="1.0" encoding="UTF-8" ?>
<<schemaxmlns=http://www.w3.org/2000/10/XMLSchemaxmlns:mpeg7="http://www.mpeg7
.org/2001/MPEG-7_Schema"
targetNamespace="http://www.mpeg7.org/2001/MPEG-7_Schema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
……
</schema>
```

声明部分包括名称空间、文档类别和版本声明，使得系统能够正确识别该文档。

文档实例部分是指具有一定结构的具体的 MPEG-7 描述内容。MPEG-7 提供了描述定义语言（DDL）来描述对象、事件、概念、状态、位置等多媒体语义信息。下面举出的例子是基于 MPEG-7 标准对一张名称为 Jim 的人物图像进行图像颜色特征描述的 XML 文档实例部分。它包含了一个 ScalableColorType 类型的描述值。

```
<image>
<imagename>Jim</ imagename >
<comments>nice picture</comments>
<tags>sunset sky</tags>
<Mpeg7>
<Description type="ContentEntityType">
<MultimediaContent type="ImageType">
<Image>
<VisualDescriptortype="ScalableColorType" numOfBitplanesDiscarded="0" numOfCoeff="64">
<Coeff>
-121 8 -3 87 12 14 22 37 31 13 11 3 50 14 19 21 -3 1 0 11
-8 5 0 17 -8 2 2 4 -15 5 1 -1 1 0 0 1 0 0 1 1 6 1 1 3 1 2 4 12
-1 0 2 2 2 3 3 -4 15 0 0 -2 1 0 -3 6
</Coeff>
</VisualDescriptor>
</Image>
</MultimediaContent>
</Description>
</Mpeg7>
</image>
```

读者阅读了上述内容，应该对 MPEG-7 标准中的颜色描述符、纹理描述符以及形状描述符有了较好的认识。接下来，向读者阐述基于内容的海量视频检索系统是如何利用 MPEG-7 标准来进行工作的。

基于内容的海量视频检索系统采用主颜色、边缘直方图以及基于轮廓的形状这三种描述符对

图像特征进行描述。具体来说，系统使用 OpenCV 图像处理库对图像抽取颜色、纹理、形状等底层视觉特征后，利用 MPEG-7 library 实现对 OpenCV 的处理结果进行描述。MPEG-7 library 是 Joanneum Research 提供的一套 C++类，使用这些类能创建并操纵多媒体内容描述对象，将程序中的多媒体描述对象序列化为 XML 文档，或者根据 XML 文档生成多媒体内容描述对象。

MPEG-7 与基于内容的海量视频检索系统的联系如图 11.25 所示。MPEG-7 处在基于内容的图像信息检索里的中心位置。MPEG-7 的输入是对图像数据分析的结果，MPEG-7 的输出则提供了图像信息提取的基础。

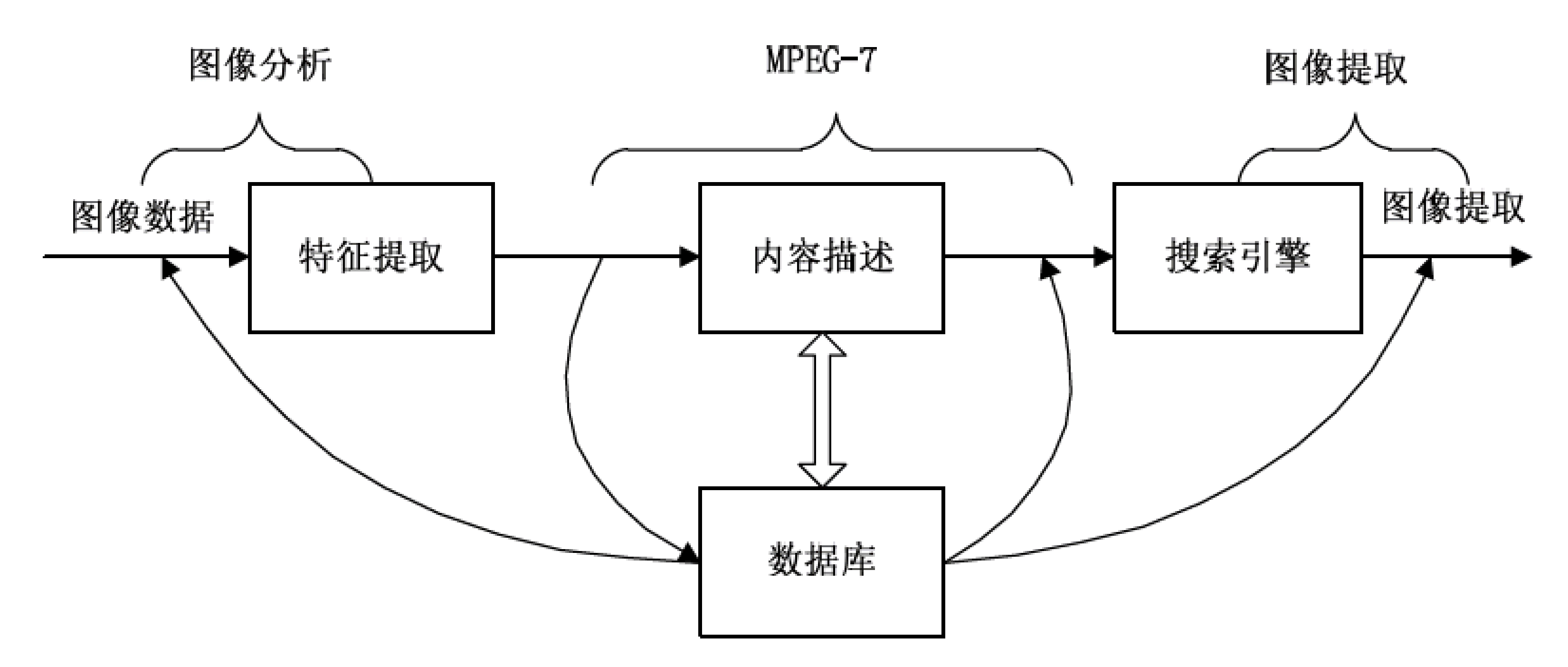


图 11.25　MPEG-7 与基于内容的海量视频检索系统的联系

具体而言，在基于内容的海量视频检索系统中，考虑到一个对象对应多张对象图片，对象的每张图片都需要进行特征提取。故将特征值存储在一个单独的 XML 文件中，它的命名规则如下：视频 ID+对象 ID+图片在原始视频中对应帧号。例如：video1_object1_5 表示视频 video1 中 object1 的第 5 帧图片。这样设计的优点如下：

- 在进行图像特征值匹配时，只需要对列族中的几列数据进行匹配，并且列族中列的数目是确定的，并且列的数目不多，XML 文件中元素路径和值之间的关系是简单的一对一关系，不需要像处理 XML 文件中具有多对多关系那样还需要在解析 XML 文件时对文件进行编码，简化了系统过程。
- 相比一个对象对应一个 XML 文件的设计方法，因为某个对象对应多少张对象图像是不可知的，需要用<object_frame>，即对象图像的帧号作为标签来分隔不同对象图片的特征值。当将 XML 文件中的内容直接解析存入 HBase 后，列族中的列的数目是不确定的，当进行特征值匹配时，假设需要对颜色：颜色空间（color/colorspace）这一属性进行匹配，则需要对列族中的多列进行匹配。
- 不需要额外对图像所属的对象以及视频名进行存储，因为匹配完毕后，可以直接根据 RowKey 得到对象名以及视频名。

XML 文件中的内容包括颜色特征、纹理特征、形状特征，XML 文件中内容的数据形式可以简化表示如下：

```
<mpeg-7>
<color>…</color>
```

```
<texture>…</texture >
<shape>…</shape>
</ mpeg-7>
```

在 HBase 的数据库设计中，可以将 XML 文件中的 path 值进行简化后设置为 HBase 中的一列。当需要对用户上传的图片进行颜色特征匹配时，只需要对由路径 mpeg-7/ color 简化后的 color 这一列数据进行匹配。在查询时，结合这三种图像特征，并对三种特征分配不同的权值，最后按照匹配度从高到低，将目标视频显示给用户浏览。

2. 基于内容的海量视频检索系统数据库设计

在基于内容的海量视频检索系统的设计过程中，对后台数据库的设计是非常重要的环节。由于基于内容的海量视频检索系统的所有数据都是存储在数据库中的，所以数据库设计合理与否将直接影响到系统的性能。结合系统的实际应用，我们既需要存储经过识别执行特定行为的海量视频数据又需要存储海量的对象图片。当系统面对海量的图片库进行图像检索以及对特定时间地点及人体进行检索时，这将会是一个非常困难的问题。因此，系统的分区容忍性和并行处理性能是非常重要的。得益于 HBase 的透明化可伸缩性和 MapReduce 的支持，同时结合从监控摄像头中获取的一段 10 分钟左右的监控视频大小约为 50MB 左右，而 HDFS 并不适合小文件的存储这个应用实际，使用 HBase 作为基于内容的海量视频检索系统的数据库是合理和高效的。

在本节中，将重点围绕基于内容的海量视频检索系统在 HBase 上的数据库设计进行说明，并对 HBase 的特点进行说明。首先对基于内容的海量视频检索系统进行需求分析。系统中的数据库模块需要与基于内容的海量视频检索系统的业务逻辑进行交互。这种交互主要包括以下几个部分：

- 视频数据交互。需要对从监控摄像头中获取的原始视频进行一系列的处理，最终获取人体行为识别模块的结果生成小视频并存储在 HBase 数据库中。当用户最后通过客户端对数据进行查询时，数据库将执行某行为的相关视频返回给用户浏览。
- 人体行为识别。需要从数据库中提取对象的灰度图片，并通过业务逻辑层中的人体行为识别算法，对对象的行为进行识别。识别完成后，将执行某一动作的一组图片在原始视频中的起始帧号和终止帧号存入数据库中，作为后期生成执行目标行为视频的依据。

根据上述交互内容，可以得到数据库系统的总体设计图，如图 11.26 所示。

数据库模块的总体设计思路是对基于内容的海量视频检索系统提供持久化的数据支持，并能够结合 MapReduce 完成数据的并行处理和分析。数据库设计方面需要能够完整地实现基于内容的海量视频检索系统的功能，并尽量提高性能。在确定数据库的总体设计思路后，进一步完成对系统中数据库表的设计。

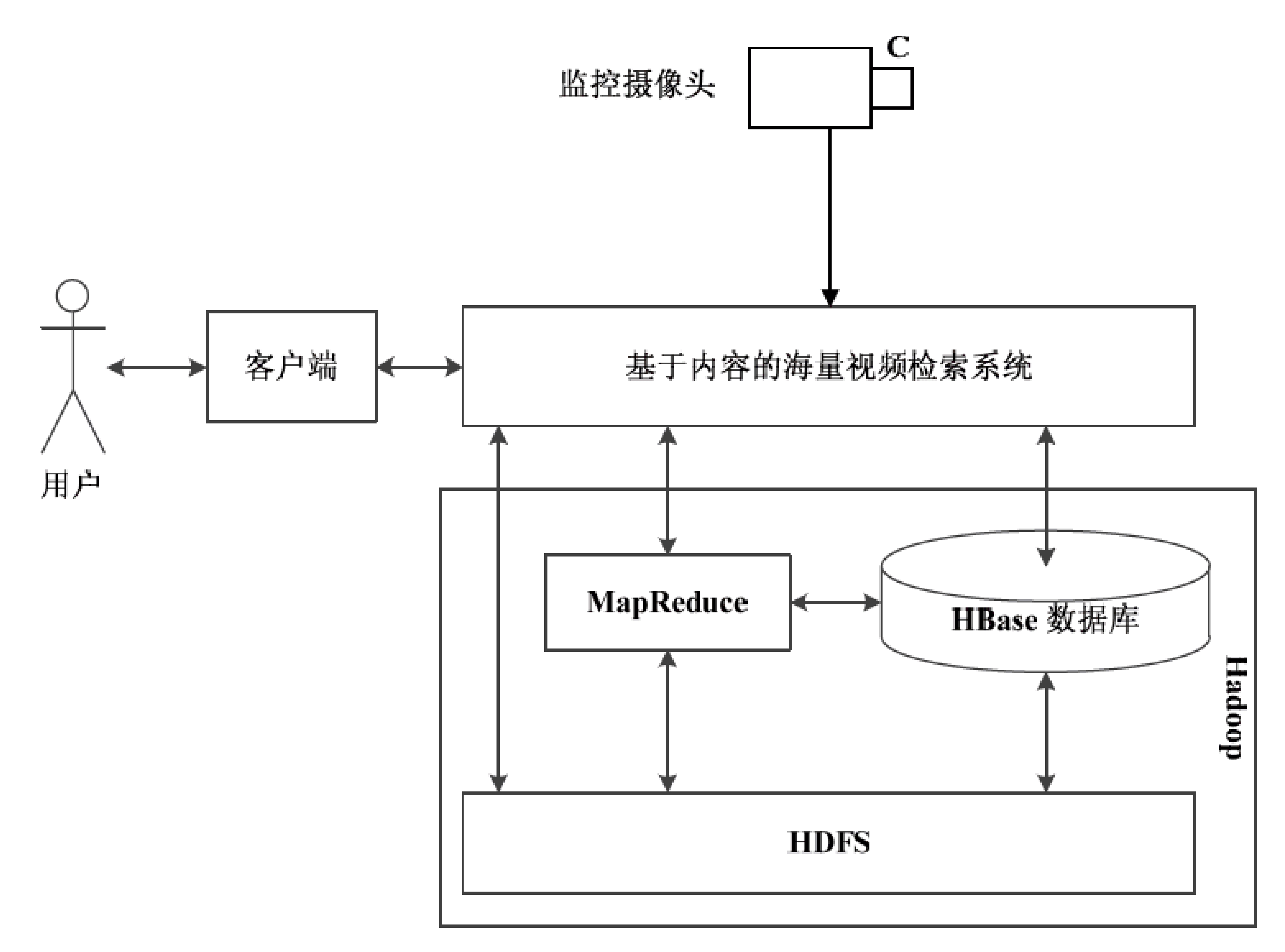


图 11.26　数据库系统的总体设计图

HBase 中没有数据关系，例如视频和对象图片这两个实体，但在 HBase 数据库的设计中仍然可以当做实体考虑。然而实体之间的关系不能通过类似于关系型数据库中的外键等约束来保证，需要按照 HBase 的特点来对这种关系进行体现。

通过对系统进行分析，可以得到 4 个实体：用户、原始视频、对象图片和视频片段。用户是系统对外提供功能的使用者，用户具有对系统进行查询的能力。原始视频是从监控摄像头中直接获取的长约 10 分钟的视频，以视频 ID 作为划分，包含了视频数据以及视频的记录时间和记录的事件地点这些信息，以供用户后期的检索使用。对象图片是系统对原始视频进行处理后得到的包含运动物体的图片，主要用于用户对视频片段的检索。视频片段是系统最终需要展示给用户浏览的信息，它是特定对象执行某些特定行为的视频。

如果是在关系型数据库中，按照数据关系模型，对于这些实体，一个用户可以查看多段视频片段，每段视频片段可以被多个用户查看，所以用户和视频片段之间形成一种多对多的关系。对于原始视频与对象图片，一段原始视频可以拥有一到多张对象图片，由于在系统的处理中，出现在不同视频中的同一对象会被处理成不同对象，假设张三出现在视频 1 和视频 2，最后张三可能会被处理成视频 1 中的对象 1 和视频 2 中的对象 5，即每张对象图片只对应着一段原始视频，所以视频和对像图片之间形成一种一对多的关系。对于原始视频和视频片段，一段原始视频可以对应多个视频片段，而视频片段只对应一段原始视频，故两者是一对多的关系。而视频片段与对象图片同样是一对多的关系，视频片段是由对象图片组合成的。于是在传统的 RDBMS 数据库设计中，会得到如图 11.27 所示的 ER 图。

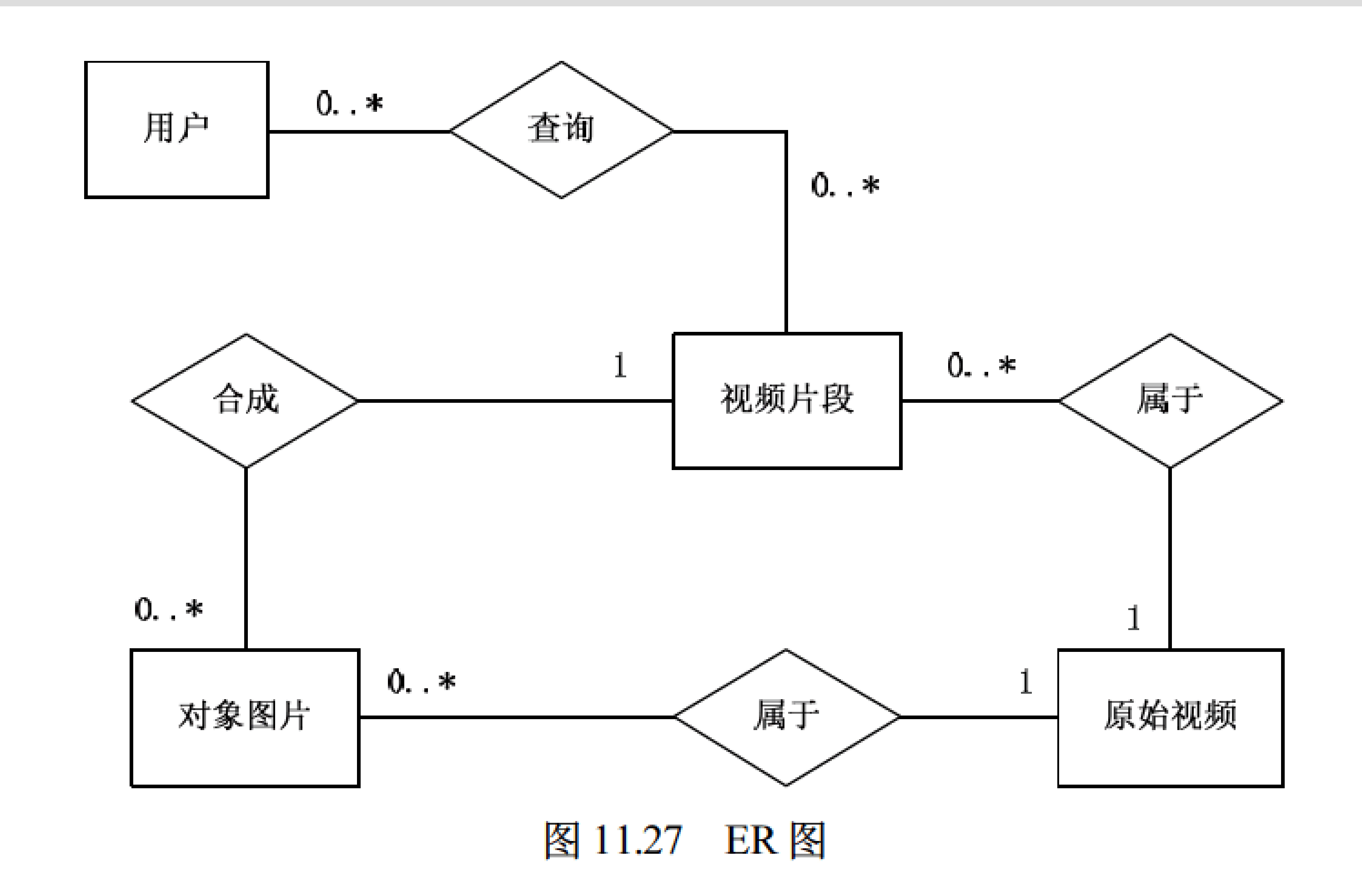


图 11.27　ER 图

可以根据 ER 图得到关系型数据库中的表格设计，如图 11.28 所示。

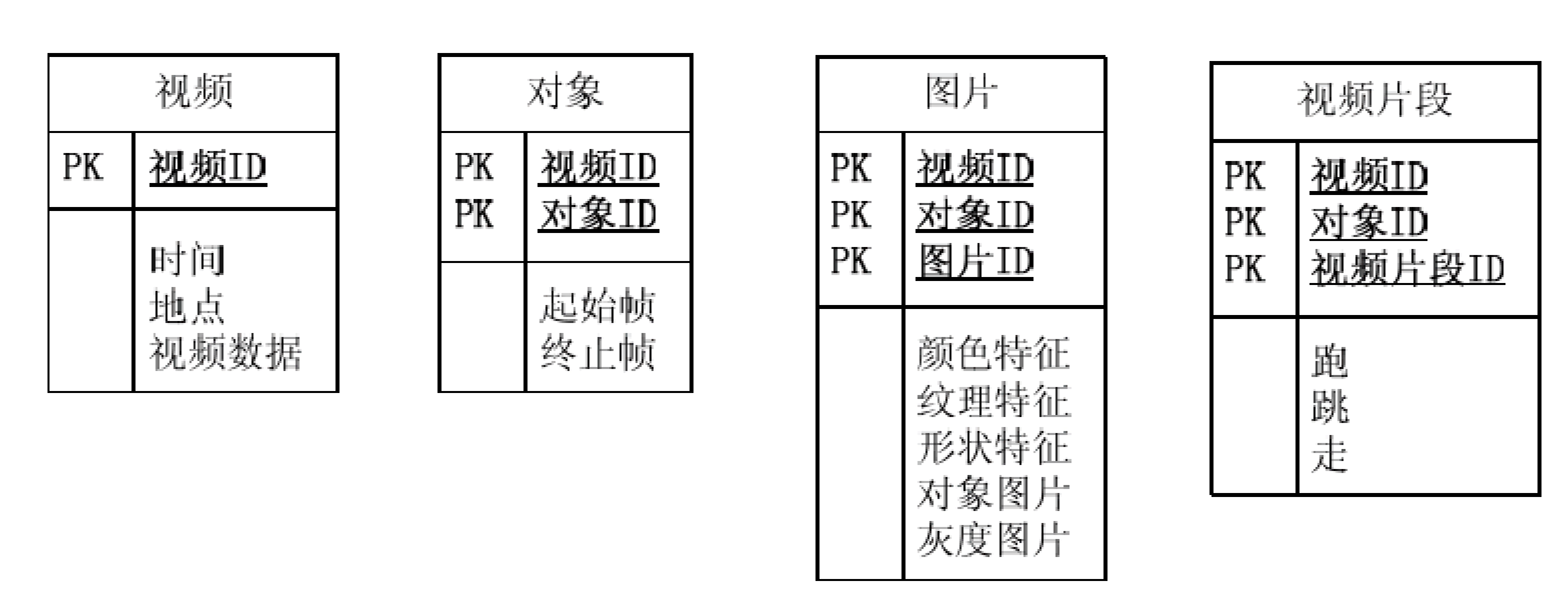


图 11.28　数据库表格设计

这样将一对多或者多对多关系分裂成多张表的方式，在关系数据库中往往能达到更高的范式，具有减少冗余、增强一致性并保证完整性的作用。然而在 NoSQL 数据库中，由于没有关系的约束，此处必须考虑到下面将要提到的 HBase 的特点来设计合适的数据库模型。

原始视频与对象图片之间的关系是一对多的关系，一段视频中存在多个对象。基于用户可能存在观看原始视频数据的需要，将原始视频存储在单独的表中，并在该表中存储视频发生的时间段以及监控摄像头所处的地点，以供后期用户查询使用。而一个对象对应多张图片，且对象图片的数目不定。考虑到对象图片的特征提取时，假设一个对象对应 N 张图片，则需要对这 N 张图片都进行特征提取。可以结合视频名称、对象名称以及图片的帧号作为图像表中的关键字，将图片的特征描述项作为同一列族中的一列进行存储。

原始视频与视频片段是一对多的关系。在进行数据库设计时，可以将视频片段存入一张单独的表中，行关键字设置为与视频 ID 相关，同一视频对应多段视频片段。

视频片段与对象图片之间的关系也是一对多的关系，视频片段由多张对象图片组成。在数据库设计中，可以将对象图片存储在一张图片表中，视频片段名中通过与该对象的起始帧和终止帧关联，来体现这一关系。

在数据表的设计过程中，除了需要注意以上内容，还需要结合 HBase 数据库的特点进行设计。HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，

与 Hadoop 一样，Hbase 目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加系统的计算能力和存储能力。在进行数据库设计时需要重点关注 HBase 的以下特点：

（1）HBase 拥有较为简单的数据模型。由于不存在关系约束等，所以在数据库表的设计上应该简单些。

（2）HBase 面向列存储的特点。读取同列数据往往具有更高的性能，在设计中应该考虑列的划分。

（3）HBase 中空单元格不占空间。其实这个特点是面向列存储的结果，这使得读者可以不必像设计关系数据库那样，将一对多和多对多关系分裂成多张表来存储，可以直接将数据放入一张大表中，省去空单元格。

（4）HBase 的列是可以动态添加的。没有固定的模式使得 HBase 数据库设计的灵活性大增，客户端可以随时增加新的列，改变现有的模式。

（5）HBase 只支持通过 RowKey 查询，访问 HBase 表中的行只有三种方式：通过单个 RowKey 访问、通过 RowKey 的 range 及进行全表扫描。要尽量遵循 HBase 的行关键字（RowKey）设计原则，具体如下：

首先是 RowKey 的长度原则。行关键字可以是任意字符串（最大长度是 64KB，实际应用中长度一般为 10~100B），在 HBase 内部，RowKey 保存为字节数组。在原则上 RowKey 应该越短越好，最好不要超过 16 个字节，原因一是数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 RowKey 过长，比如 100 个字节，1000 万列数据只计算 RowKey 就要占用 100*1000 万=10 亿个字节，将近 1GB 数据，这会极大影响 HFile 的存储效率；原因二是 MemStore 将缓存部分数据到内存，如果 RowKey 字段过长，内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。因此 RowKey 的字节长度越短越好。原因三是目前操作系统都是 64 位系统，内存 8 字节对齐。控制在 16 个字节，8 字节的整数倍利用操作系统的最佳特性。

其次是 RowKey 的散列原则，如果 RowKey 是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将 RowKey 的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个 RegionServer 实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个 RegionServer 上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别 RegionServer，降低查询效率。

最后是 RowKey 唯一原则，必须在设计上保证其唯一性。

在遵循上述原则的同时，在设计 RowKey 时，还需充分利用 RowKey 排序存储的特性，将经常一起读取的行存储放到一起，这样可以大幅度提高数据库查找性能。

（6）HBase 中的更新操作都对应着一个时间戳。在设计时可以适当地考虑通过时间戳来记录数据的版本号，或者对数据进行备份等。

根据基于内容的海量视频检索系统实体间的关系，结合利用 HBase 的特点，对系统进行数据库设计以满足系统的需求。具体而言，基于内容的海量视频检索系统的数据库共包括三张表，分别为视频表、图片表和视频片段表。表 11.6~11.8 为 HBase 中的数据库表设计。

（1）视频表（video_infor）的设计

- 列和列族：视频表主要用来存储与原始视频相关的信息，将视频发生的时间、视频内容记录的地点基于原始视频数据存入列族 video_infor 中。当用户通过客户端进行检索时，需要对事件时间和事件发生地点这两个条件进行筛选。此时直接读取视频表中 video_time 和 video_place 这两列中的特定 RowKey 范围内的数据进行匹配。当用户获取满意相关视频片段后，还可以查看视频表中的原始视频数据作为进一步的参考。
- 行关键字：将行关键字设置为视频的 ID 号，一行数据保存一段视频的基本信息。视频 ID 号在后面将要讨论的图片表以及视频片段表中都有应用，具体的内容请阅读下面的内容。

表 11.6　视频表设计

行关键字 RowKey	列族 Column family： video_infor	
视频 ID	列标签	备注
	video_time	视频时间
	video_place	视频地点
	video	原始视频数据

（2）图片表（picture）的设计

- 列和列族：图片表主要记录对象的原始图片以及图片的各种信息。由于系统需要按照 CBIR 算法来检索图片，所以需要预先对图片的内容特征信息进行提取，并存入数据库中，即将颜色、形状和纹理等特征值存放在列族 picture 中。HBase 存放图像特征具有较好的可扩展性，对于后期的检索特征算法修改具有很好的兼容性。与此同时，还需要将对象图片和对象灰度图片存放入列族 picture 中。对象图片用于提取行为识别模块中的识别结果生成对象表中的执行特定动作的视频片段， 即显示给用户的视频由它组合成。而对象灰度图则是作为人体行为识别模块的输入。
- 行关键字：考虑到一个对象对应多张对象图片，且在之前对所有对象图片进行特征值提取并写入 XML 文件时，一个对象的每张图片都对应一个单独的 XML 文件。利用 HBase 行关键字的有序特性，可以将对象图片的行关键字直接设计为 XML 文件的名字，即“视频 ID+对象 ID+图片在原始视频中对应帧号”，有关具体的实例，读者可以参考上一部分的内容。

表 11.7　图片表设计

行关键字 RowKey	列族 Column family： Picture	
对象图片 ID	列标签	备注
	object_picture	对象图片
	gray_picture	对象灰度图
	color	颜色特征
	texture	纹理特征
	shape	形状特征

（3）视频片段表（video_segment）的设计

- 列和列族：视频片段表主要记录对象执行某些行为的视频片段信息。考虑到人物对象的行为数目是有限的，将对象执行走、跑、跳等动作的视频片段设置为列族 video_segment 中的一列，方便基于内容的海量视频检索系统在行为识别模块上的拓展性，例如现在系统可以识别打架这一行为，只需要在列族中再添加 fight_segment 即可。在视频片段表的设计中，具体而言，run_segment、jump_segment、walk_segment 这三项是用于记录对象执行相应动作的视频片段，该列中存储的视频名称为“视频 ID+对象 ID+起始帧号+终止帧号”。注意，这里的帧号指的是在原始视频中的帧号。
- 行关键字：考虑到对象特定动作下的视频片段可能是多段，例如：同一对象在一段时间内执行跑的动作的视频片段就有可能是多段。将行关键字设置为“视频 ID+对象 ID+片段 ID”，对同一对象，片段 ID 按从 1 到 N 的升序排列，即一个视频片段一行。因为同一个对象执行各种动作的视频片段数目不一样，所以采用上述方式设置关键字的好处还在于便于统计特定对象执行各种动作的视频数目。例如：视频 1 的 ID 为 video01，其中对象 1 的 ID 为 object01，一共有 3 段跑步的视频片段，有 7 段跳远的视频片段，有 5 段走路的视频片段，那么在 HBase 中，RowKey 就为 video1_object1_1~video1_object1_7 这样的连续 7 行。在统计该对象走路的视频片段数目时，可以调用 HBase 提供的 API，获取列族 video_segment 下列 walk_segment 的非空项数目，也就是视频 1 中的对象 1 走路视频片段的数目。在 HBase 中，空的单元格并不占据存储空间，这也为上述的设计方式提供了支持。

表 11.8　视频片段表设计

行关键字 RowKey	列族 Column family：video_segment	
对象视频片段 ID	列标签	备注
	run_segment	对象跑步的视频片段
	jump_segment	对象跳远的视频片段
	walk_segment	对象走路的视频片段

3. 数据库模块交互设计

基于内容的海量视频检索系统中存在着视频表、图片表和视频片段表这三张表，那么这些表是如何配合系统的业务逻辑模块完成系统功能呢？本小节围绕数据库模块与系统业务逻辑的交互设计进行说明。

（1）视频片段搜索交互

视频片段的的搜索过程如下：首先将用户选择的事件时间、事件地点与视频表中的相关数据进行匹配，得到视频 ID。接着结合视频 ID，将用户上传图片的特征信息与数据库图片表中的特征信息进行对比得到对象图片 ID，并获取视频 ID 和对象 ID。最后再根据用户选择的人体行为名

称以及由命中结果集得到的视频 ID 和对象 ID 到视频片段表中进行查询，最后获取命中的视频片段内容并展示给用户的客户端。整个交互过程如图 11.29 所示。

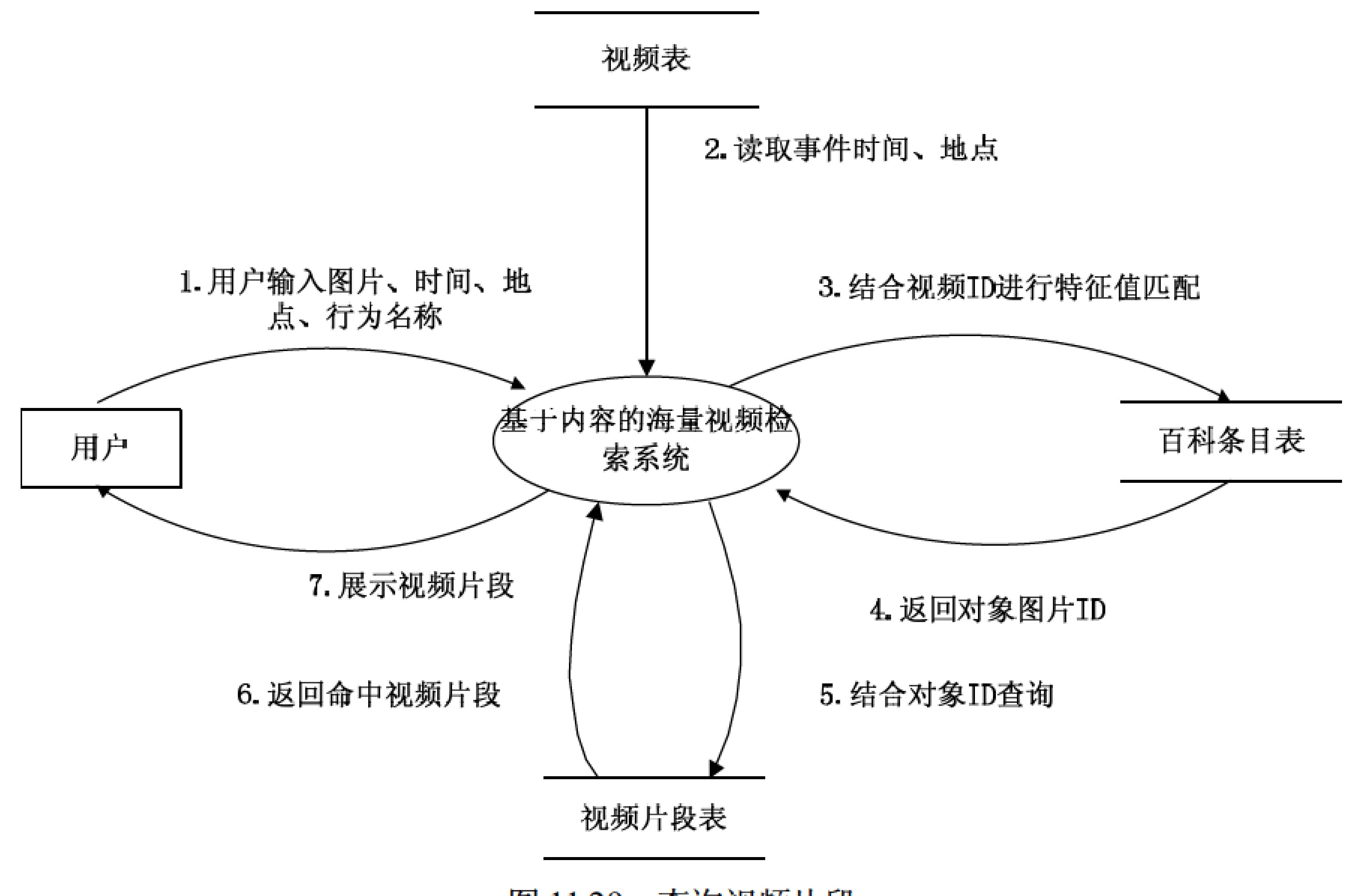


图 11.29　查询视频片段

（2）行为识别数据交互

基于内容的海量视频检索系统行为识别模块的数据与数据库的交互也非常频繁，详细过程如下：首先，需要读取图片表中的对象灰度图，并得到对象图片 ID；然后行为识别模块对对象灰度进行处理，识别出人体的行为，结合对象图片 ID 生成执行对应动作的视频片段，并将内容写入视频片段表中。整个交互过程如图 11.30 所示。

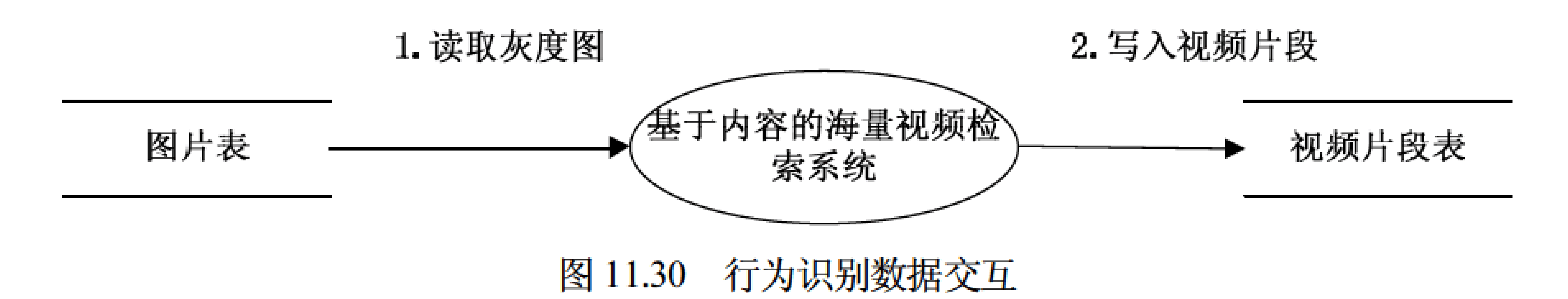


图 11.30　行为识别数据交互

本节中的交互操作围绕着 HBase 中的表，并没有涉及 HDFS 和 MapReduce 的相关交互，有关 HBase 和 HDFS 与 MapReduce 间的操作，可以阅读本书的相关内容。

4. 基于内容的海量视频检索系统的数据库实现

在完成了数据库的设计后，接下来需要对它进行具体实现。系统采用 Java API，利用 Eclipse 进行客户端编程实现。接下来主要介绍数据库的模块类的交互图以及核心类的实现过程。

由于在基于内容的海量视频检索系统的数据库设计中，有视频表、图片表和视频片段表三张不同的表，而对于每张表需要提供的方法都不尽相同，因此，对于每张表提供一个单独的类进行操作是有必要的，然而这三张表也具有一些公共操作，可以将其抽象为一个表的基本数据操作类，

于是形成了如图 11.31 所示的类图。

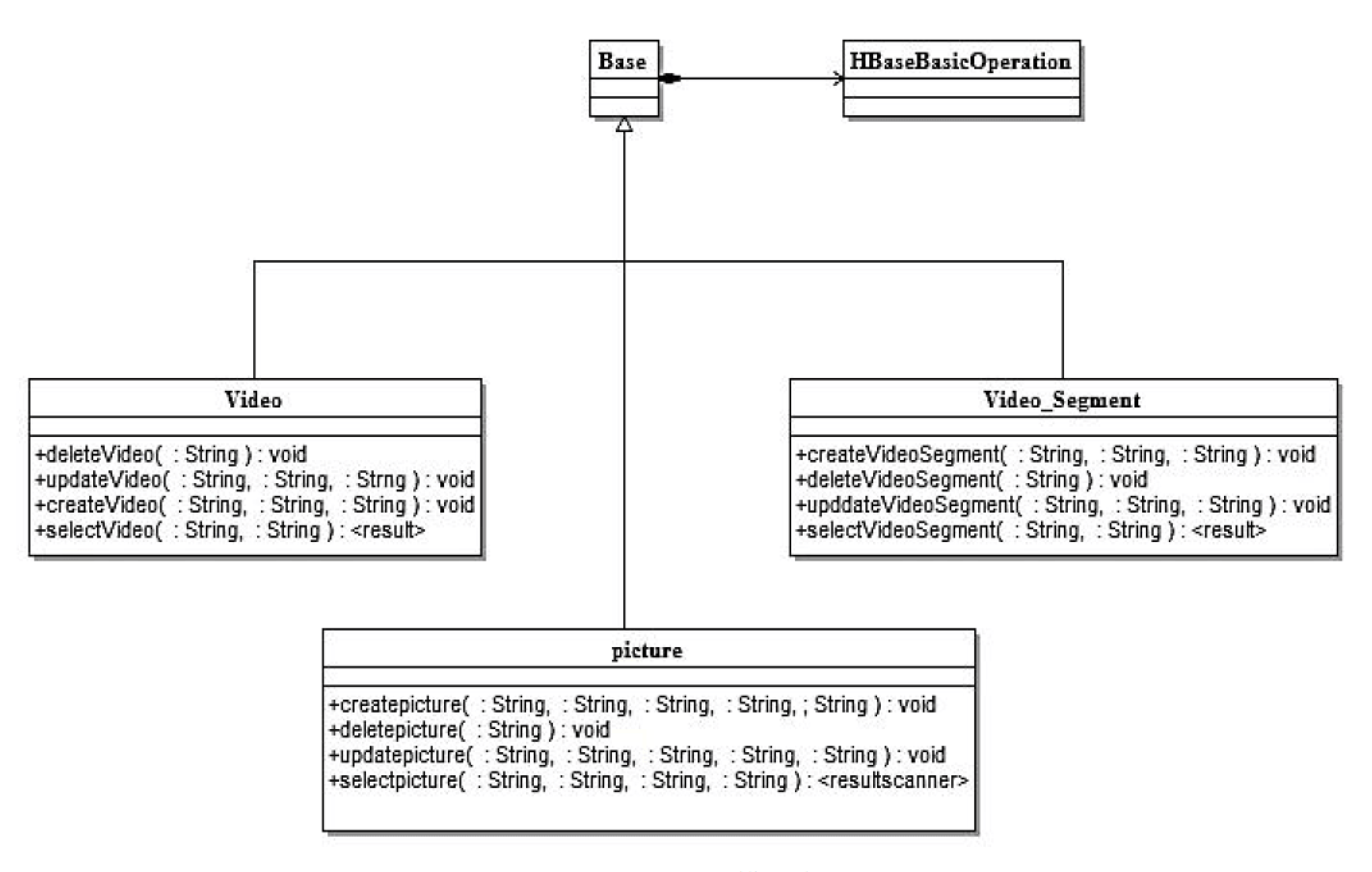


图 11.31 数据库模块类图

从图 11.31 中可以看出，数据库模块主要由 5 个类组成，其中 HBaseBasicOperation 类为 HBase 的基本操作类，该类是一个通用的操作类。其余的 Base、Video、Picture、Video_Segment4 个类是表的操作类，这些类使用了通用操作类封装好的 HBase 操作接口。下面分别介绍这 5 个类：

- HBaseBasicOperation 类：这个类是一个对 HBase 基本操作进行了封装的通用类。该类主要是利用 HBase 提供的 API，对 HBase 的配置文件等进行初始化，并实现了包括创建、插入、删除、查找等基本操作的通用方法。
- Base 类：这个类是对表操作的基本类。该类是 Video、Photo 和 Video_Segment 等类的基类，主要是抽象出了这些特定表的一些共同操作，提高代码复用率和开发效率。该类直接调用 HBaseBasicOperation 类的方法，实现了表的初始化操作和读写操作等。
- Video 类：这个类主要实现了对视频表的相关操作，包含事件时间、地点以及原始视频数据的信息。该类通过行关键字与 Photo 类相联系。
- Picture 类：该类主要实现了对图片表的相关操作，包括将用户上传的图片信息插入数据库、查询特定特征信息、获取对象图片、获取灰度图片等方法。该类通过行关键字和 Video_Segment 类关联。
- Video_Segment 类：该类主要实现了对视频片段表的相关操作。包括获取特定名称的视频片段等方法。该类通过行关键字与 Video 表类和 Photo 类进行关联。

接下来，介绍一下这些核心类的实现，由于篇幅有限，在这里只做简要说明。

（1）HBaseBasicOperation 类

作为通用的 HBase 操作类，该类中是直接采用 HBase API 对数据库进行操作的。该类首先完成配置初始化，设置一些配置信息，并在初始化按照设置参数获得一个 HBase 配置实例 conf。完成初始化配置后，便可以对 HBase 进行操作。要对 HBase 进行管理，例如表的创建和删除、列出表清单以及表结构更改等，则使用 HBaseAdmin 类。一旦表被创建，则需要通过 HTable 类的实例来操作该表。每次可以增加一行内容到一个表中。对于插入操作，则需要创建一个 Put 类的对象实例，并指明目标列名（target column）、值（value）以及一个可选的时间戳，最后使用 HTable.put（Put）方法执行插入操作。要获取表中已插入的值，则使用 Get 类，Get 类的实例可以被指定为获取指定行的所有内容或者仅仅获取该行中的某一个单元格的值。在创建了 Get 的一个实例后，调用 HTable.get（Get）执行查询，返回结果是一个 Result 类的实例。还可以使用 Scan 类建立一个扫描器（scanner），这类似于数据库中的游标。在创建并配置了 Scan 类的实例后，则调用 HTable.getScanner（Scan）获取一个 ResultScanner 类的实例，该实例就是查询得到的结果集，ResultScanner 事实上是 Result 实例的一个集合，循环调用 ResultScanner.next()方法可以获取每个 Result 实例。一个 Result 是一组 KeyValue 的集合，它具有将不同类型的返回值进行打包的功能。使用 Delete 类来删除表中的内容，用户可以设定 Delete 类的实例用于删除一个独立的单元格或者整个列族，并且将该实例传递给 HTable.delete（Delete）来执行。

（2）Base 类

Base 类是对表进行操作的基本类，该类主要通过 HBaseBasicOperation 类中提供的方法实现各个表中的公共操作。首先，Base 中存在一个 String 类型的变量 tableName，在子类继承时，应该首先通过构造方法对该变量进行赋值，以指明子类操作的表名。所以该类提供了一个带参数的构造方法 Base（String tableName）用以实现该需求。接下来简要叙述 Base 提供的一些基本操作。Base 主要实现增删改查的操作，createTable 是用来创建一个名为 tableName 的表，insertRow 用于向 HBase 中插入一行信息，deleteRow 用于删除表中行关键字为 RowKey 的一列。上述方法在这里都不做详细介绍，下面主要以 selectRow 这个查询方法作为例子详细说明。

selectRow 方法用于选择表中行关键字为 RowKey 的一行信息，并返回一个 Result 结果集。

```
protected Result selectRow(String RowKey){
Result r = null;
try {
 r= HBaseBasicOperation.selectRow(tableName, RowKey);
} catch (IOException e) {
 e.printStackTrace();
 }
return r;
}
```

通过直接调用 HBaseBasicOperation 类的 selectRow(tableName,RowKey)方法，返回的结果是一个定义在 org.apache.hadoop.hbase.clent.Result 中的 Result 实例。由于不同的表对结果的解析不同，所以基本操作类中不对结果实例进行分析，而直接返回给上层具体表的操作类中。

（3）Video 类

Video 类是针对视频表的操作类，它继承了 Base 类，其基本操作主要是查询操作，即 selectVideo，实现以表名、列族名以及列名作为输入条件，获得结果。考虑到 Video 类的基本操作都可以直接调用基类中的操作直接完成，在这里不做详细介绍。

（4）Picture 类

Picture 类是针对图片表的操作类，它继承了 Base 类，并封装了需要对图片进行操作的基本方法。具体而言包括图片信息的插入、删除和更改，并且每次图像检索都需要查询数据库中的一部分图像特征，因此该类还包含了对图像信息，特别是特征信息的查找获取等方法。Picture 类除查询以外的方法与 Video 类类似，所以下面主要介绍 Picture 类查询数据的方法 selectPhoto。

此方法主要实现获取特定行关键字范围内的特定列族中某一列数据的多行信息，并返回一个 ResultScanner 结果集。

```
   protected  ResultScanner  selectPhoto(String  tableName,String  RowKey,String
column,String qualifier){
   try {
       HTable table=new HTable(conf,tableName);
       Scan scan = new Scan();
       scan .addColumn(Bytes.toBytes(column), Bytes.toBytes(qualifier));
      scan.setFilter(new PrefixFilter(RowKey.getBytes());
     ResultScanner rs= table.getScanner(scan);
   }catch (IOException e)
   {
       e.printStackTrace();
    }
    return rs;
   }
```

此处使用了 PrefixFilter 这种过滤器，PrefixFilter 是行关键字的前缀过滤器，这个过滤器的主要功能是命中所有以特定前缀开头的行关键字所在的行。例如前缀是 video01_object_01，那么例如行关键字为 video01_object01_01，你好 video01_object01_19 等行会被命中，而 video01_object02_01、video02_object01_19 等不是以该前缀开头的行被过滤掉。

PrefixFilter 的构造方法为：PrefixFilter（final byte [] prefix）。因此创建一个 PrefixFilter 过滤器时需要传入一个 Byte 数组格式的前缀，作为过滤条件。此外，PrefixFilter 提供了一个 getPrefix() 方法用于获取过滤器中的前缀。

（5）Video_Segment 类

Video_Segment 类也是 Base 类的子类，实现了视频片段的基本操作，由于其基本操作与之前两种类类似，此处不再赘述。

在完成各个核心类的实现后，运行程序可以创建数据库中的三个表，并结合实际需要对各表进行操作，最后实现系统的功能。

11.4.3 行为识别与运动规则的组合创建

1. 规则创建过程

（1）规则的定义

这里所说的规则主要是指行为规则，它根据粒度的大小分为三类，分别是 pose、action 和 activity。pose 指的是一个姿势，它代表这一帧图片中某一对象的某个状态，如图 11.32 为一个 pose。action 指的是动作，它代表一种不可再细分的行为，由一组姿势组成，如走、跑、跳。activity 代表的是行为，由一系列的动作组成，如三级跳等。



图 11.32 一个 pose

本系统的设计中，pose、action 和 activity 作为三个类进行设计和实现的。类图如图 11.33 所示。其中 pose 只由一个属性 angle 组成，指的是该 pose 按照星型骨架方法提取的行为特征值以“|”分割，拼接组成的字符串，如 angle=θ1|θ2|θ3|θ4|θ5|xc|yc。action 包含 name、num、feature、v 和 a 共 5 个属性。name 代表 action 的名字，如走；num 代表这个 action 规则是由几帧图片组成的；feature 代表行为特征值，类似于将多个 angle 以“|”分割，拼接起来；v、a 分别表示此规则中行为的速度以及加速度。activity 的类定义与 action 相同，只是没有速度和加速度的定义，这里不再详细描述。

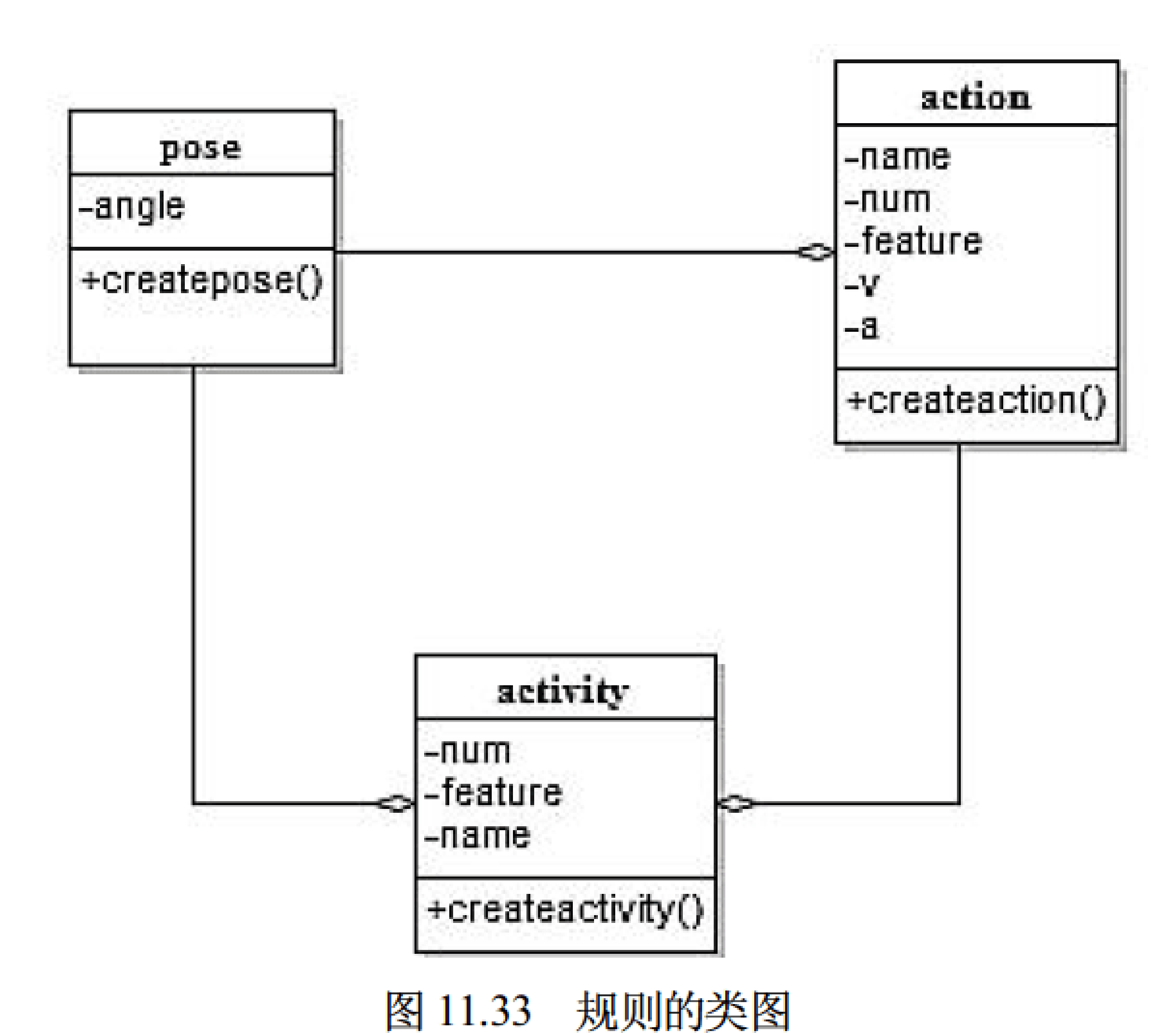


图 11.33 规则的类图

（2）创建规则

创建规则分为三类，分别是 pose 规则创建、action 规则创建以及 activity 规则创建。下面详

细描述 pose 规则、action 规则、activity 规则的创建过程。

- pose 规则创建

对于 pose 规则特征的创建，输入 pose 对应的图片路径，再按照前面所述的星型骨架方法提取行为特征值，然后将行为特征值转化为字符串型并拼接起来，形成一个新的字符串变量赋给 pose 类的 angle 变量即可。pose 规则创建的流程图如图 11.34 所示。

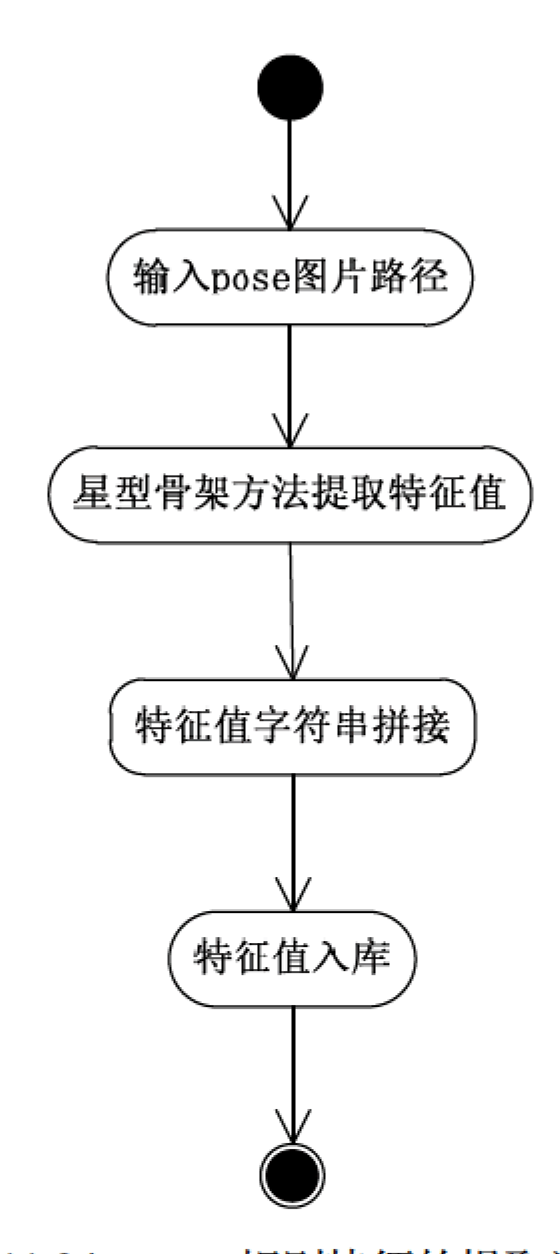


图 11.34　pose 规则特征的提取流程

pose 规则特征的提取过程的伪代码如下。

```
Begin
输入图片路径 s
c[M]←{0.}
b[N1][N2]← {0.}
id[K]← {'0'}
length← 0
length← LunKuo(b,s)                //提取轮廓特征
guize(b,length/2,c)                //提取角度特征
for i← 0 to M
     dogcvt(c[i],sig,str)
//gcvt 函数将浮点数转化成字符串并返回指向字符指针，sig 表示存储的有效数字位数
p.angle← p.angle+str+"|"           //p 为 pose 类型，角度特征之间用"|"分开
repeat
itoa( num,   id,  10 )             //将整型转化为字符型
video_insert("pose",id,s,p.angle)  //将新建的 pose 存入数据库
return p
End
```

- action 与 activity 规则的创建

对于 action 以及 activity 规则的创建，与 pose 类似，即输入路径，然后按照星型骨架方法对每帧图片进行处理，获取每帧图片的行为特征值，将所有行为特征值拼接起来作为 action 或者 activity 的特征。但不同的是，涉及训练的过程，因此输入的图片文件夹里，可能包含几组该行为过程的图片，对每组进行处理，然后求取平均值，作为这个规则的特征。action 的创建过程的流程图如图 11.35 所示。

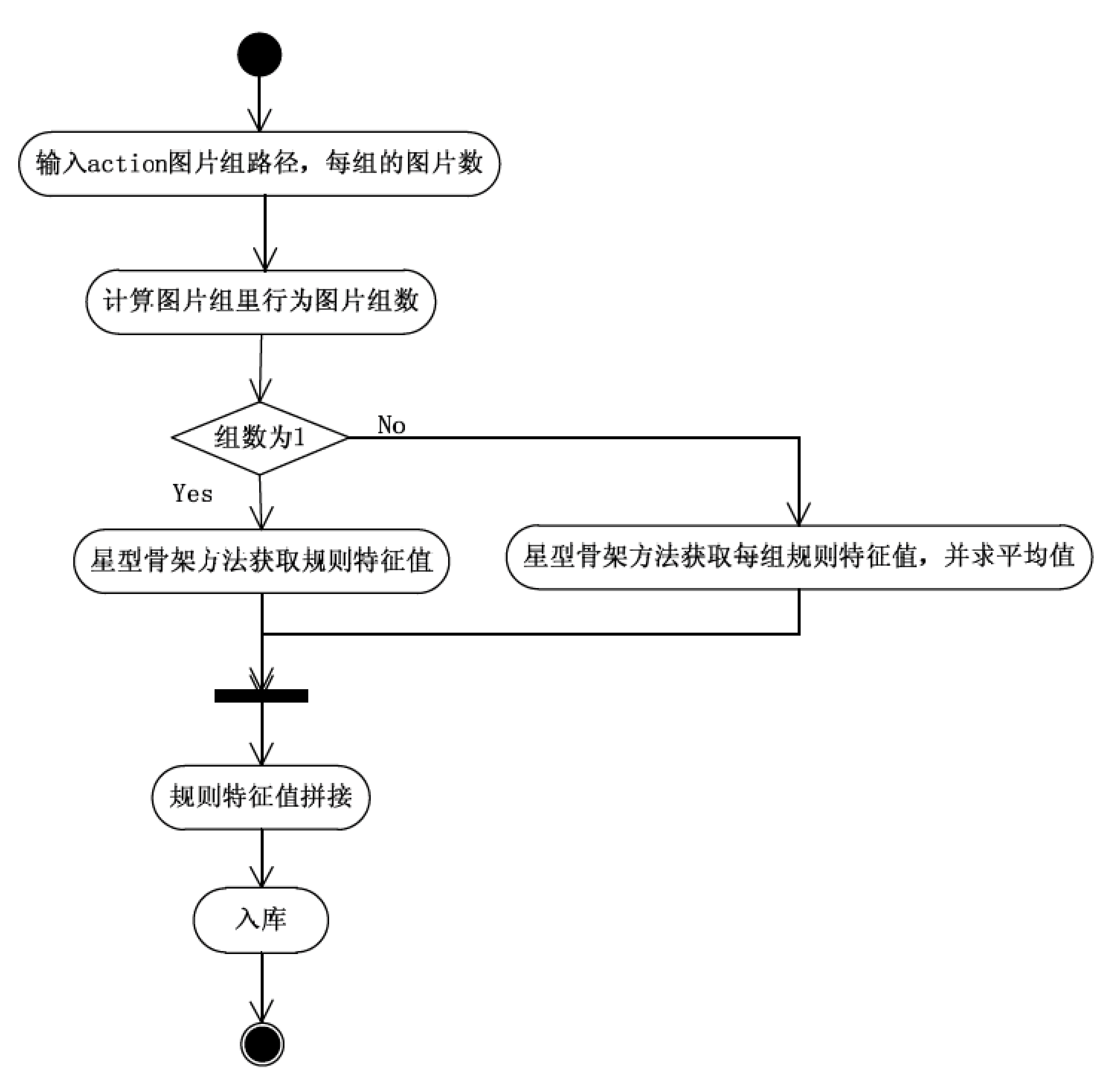


图 11.35　action 规则创建流程图

action 规则创建的伪代码如下。

```
Begin
输入文件路径 s，描述一个 action 所需的图片数 n
po[Max]                          //假定一个 action 最多由 Max 个 pose 组成
FileName[M] ← {""}               //存储文件名
j ← k ← 0
GetFileName(s)                   //获取文件夹下的文件名
For i ← 0 to total               //total 为文件夹下的文件
po[i] ← create_pose(FileName[i]) //创建文件夹下图片的 pose
Repeat
IF num == n                      //num 为一个文件夹中的文件数目
则
For i ← 0 to num                 //只输入了一组 pose，对 pose 进行拼接形成 action

act.feature ← act.feature + po[i].angle   //act 为 action 类的对象
act.n ← n
Repeat
```

```
否则
m ← num/n
for i ← 0 to num
IF k == m
则  k ← 0
j ← j+1
否则                    //ac[]为一个 action 对象组成的数组
ac[j].feature ← ac[j].feature + po[i].angle  //求出 m 组的 action
k ← k+1
Repeat
For i ← 0 to m
split(ac[i].feature,'|',v)     //分割特征值,并存储到数组 v 中
vf[it] ← vf[it] + v[it]         //把每组 action 对应的特征值相加
Repeat
gcvt(vf[it],4,str)
//将浮点数转化成字符串，并返回指向字符指针，sig 表示存储的有效数字位数
ac1.feature+=s+"|" //把最终的特征值拼接起来
End
```

activity 规则与 action 规则的创建过程类似，此处就不详细描述了。

2. 行为识别

本系统对行为识别采用的是预处理的方式。即对于任意一段要搜索的视频，采用预处理的方式，将整个视频中发生的行为进行预先识别，在真正搜索的过程中，因为行为用文本表示，所以搜索时只进行文本匹配即可。

在行为识别的过程中，因为 pose 很难代表一种行为，所以行为识别的过程主要是对视频中的 action 以及 activity 的识别。因为 action 与 activity 规则是由单对象的一组行为图片组成的，而一张行为图片对应一个 pose。所以 action 与 activity 的识别是以 pose 识别为基础的。在此，先介绍一下 pose 的匹配。

（1）pose 匹配

对于 pose 匹配，实际上是两个 pose 的行为特征，按照相似度匹配公式进行匹配。但是由于是对角度值进行相似度计算，角度的变化范围最大是（0，360），差值太小，需要进行一定的处理。经实验验证，对其的更改是减去 0.9，再乘以 10。如若结果值为负值，将结果值变为 0。本系统实际上 pose 的相似度计算公式如下。经实验验证，当相似度大于 93%时，认为其相似。

$$\text{Similarity}\left(\text{pose}_i,\ \text{pose}_j\right)=\text{Similarity}\left(\theta_i,\ \theta_j\right)=\left(\frac{\sum_{m=1}^{5}\theta_{im}\cdot\theta_{jm}}{\sqrt{\sum_{i=1}^{5}\theta_{im}^{2}}\cdot\sqrt{\sum_{i=1}^{5}\theta_{jm}^{2}}}-0.9\right)\cdot 10 \qquad 式（11\text{-}22）$$

（2）action 以及 activity 行为识别

在进行行为匹配的过程中，实际上可以理解成是 action 规则与要搜索的视频的匹配。action 实际上是由按顺序排列的 pose 组成，视频也是由一组 pose 组成。所以 action 与视频的匹配实际上是一组 pose 与另一组 pose 的匹配。它的匹配按照下面的步骤进行：

步骤 01 对于 action 规则的第一个 pose，与视频 L 的第一帧按照 pose 匹配规则匹配。若成功，则进行第（2）步，若不成功，则进行第（3）步。

步骤 02 action 规则的当前 pose 与视频 L 的下一帧匹配，若成功，重复第（2）步。直到 action 的所有组成 pose 匹配完。若不成功，进入第（3）步。

步骤 03 在 action 规则的当前帧与视频的前一帧匹配，与视频当前帧不匹配的情况下，将 action 规则的下一帧与视频的当前帧进行匹配，若成功，则进行第（2）步，若不成功，进入第（4）步。

步骤 04 action 的当前 pose 与视频 L 的下一帧 pose 进行匹配。若成功，进入第（2）步。否则，则先判断视频中当前帧与前一个匹配成功的帧之间隔的帧数，如若隔的帧数小于 4 帧，则进行第（4）步。否则，重新匹配。action 的第一个 pose 与跟 action 第一帧匹配的视频帧的下一帧进行重新匹配。

步骤 05 如果 action 的 pose 与视频的最后一帧都未匹配上，则匹配失败。

action 匹配的流程图如图 11.36 所示。

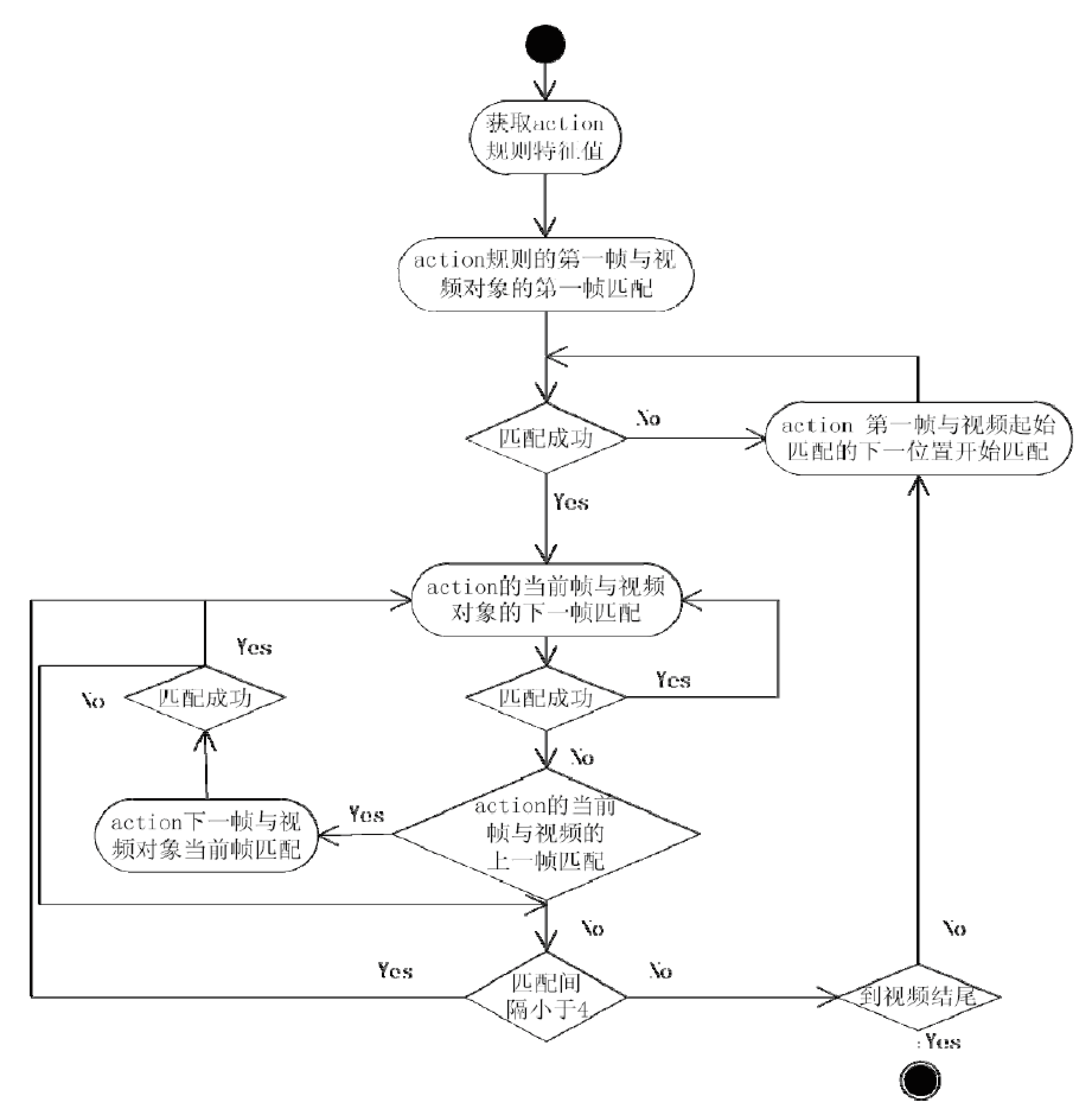


图 11.36 action 匹配过程

activity 的匹配过程与 action 的匹配过程类似，此处就不详细描述了。

接下来在行为识别过程中，首先需要声明的是，所有的动作、行为按照质心的变化预先分好类，根据质心的变化情况，将整段视频划分成几段。每一段根据质心的变化，与对应的一些动作或者行为特征进行一一匹配，按照上述匹配的规则进行，找出对应的动作或者行为，用文本标记出它所属的动作。如果找不出，同样反馈出来，供用户创建新规则。其中在与对应的动作或行为进行匹配的过程中，如果与动作或行为对应的两帧之间的间隔大于 4 帧，则不认为是匹配成功，需重新匹配。

在 action 或者 activity 与视频匹配的过程中，由于一段视频中可能包含多个 action 或者 activity，因此在匹配时标记出视频中与 action 的第一个 pose 匹配成功的帧。若匹配成功，则再记录下 action 或者 activity 的最后一个 pose 匹配的帧的帧号，以及匹配结果所属的类别，以便用于搜索；若不成功则忽略已经记录下的第一个 pose 匹配成功的帧号。

11.5 系统运行时截图

本系统采用 JSP 技术实现系统的前台交互功能，用户可以按照需求输入时间、行为、人物进行相关视频的检索，图 11.37 为系统输入查询条件的主界面。



图 11.37　视频查询主界面

在本系统中，用户可以通过下拉菜单进行行为的选择，通过“上传”按钮实现指定人物的查询，为了提高查询的准确率，将会输出一组相似性最高的视频片段，查询结果如图 11.38 所示。用户可以通过双击感兴趣的视频片段对其进行播放观看，如图 11.39 所示，播放控件允许用户对视频进行“快进”、“快退”、“暂停”等操作。

video_segments/01_jump.avi_OBJ0_run_0.avi

video_segments/01_jump.avi_OBJ0_run_17.avi

video_segments/02_jump.avi_OBJ0_run_0.avi

video_segments/02_jump.avi_OBJ0_run_18.avi

video_segments/02_jump.avi_OBJ0_run_36.avi

共有5条记录

图 11.38　查询结果显示



图 11.39　播放视频片段

11.6 本章小结

本章对海量监控视频检索系统的设计与实现过程进行了详细的介绍。首先介绍了该系统的应用背景，通过输入时间、人物、地点等条件从海量视频中检索出所需内容。然后分别从功能需求和非功能性需求两方面对系统进行了需求分析，并给出了系统中每个核心模块的业务处理流程以及系统的总体设计。第三节着重介绍了本系统所涉及的相关技术，使读者能够更好地理解本系统的设计。第四节详细介绍了系统每个模块的设计与实现，通过 HBase 进行信息的存储，通过 MapReduce 进行视频的预处理以及基于规则的行为识别与规则创建。

本章通过对海量监控视频检索系统从设计到实现的详细描述，能够让读者对于大数据在视频监控方面的应用有更好的理解。

第 12 章 基于HDFS的云文件系统

本章全面介绍了一个基于 HDFS（Hadoop Distributed File System）的云文件系统，为用户进行数据存储提供支持。传统的网盘技术具有传输速度慢、容灾备份及恢复能力低、安全性差、营运成本高等瓶颈，本系统是对开源云计算架构 Hadoop 的分布式文件系统 HDFS 进行的定制与改造，实现了面向高速局域网网络服务的云计算分布式文件系统，并提供了网盘应用的主要功能。

12.1 应用背景介绍

我们生活在数据的时代，很难估计全球以电子方式存储的数据总量有多少。数据总量的增加使得单一的本地物理小容量存储已经无法满足信息化社会的快速发展，远程云端存储正显现出超常的优越性和迫切的需要性。越来越多的网络公司和企业已经推出了网盘应用，向用户提供文件的存储、访问、备份、共享等文件管理功能，网盘类似于一个放在网络上的硬盘或 U 盘，不管在家中、单位或其他任何地方，只要连接到因特网，就可以管理、编辑网盘里的文件。但是随着存储需求和网络的进步，传统网盘技术出现了种种问题，严重制约了网络存储的发展。

最新应用的云计算储存技术，为网盘行业带来了新的革命，相对传统网盘技术它的众多优点逐渐显现。云存储是构建在高速分布式存储网络上的数据中心，它可以将网络中大量不同类型的存储设备通过应用软件集合起来协同工作，提供面向网络服务的云计算分布式文件系统。它是一个高度容错性的系统，适合部署在廉价机器上，可以提供高吞吐量的数据访问，非常适合大规模数据集上的使用，它可以从整体上提高数据带宽。系统具有分布式存储与高可用性，能够为用户提供基于云文件系统上的文件的创建、存储、删除、查询、副本设置等功能；形成一个安全的数据存储和访问的系统，适用于各大中小型企业与个人用户的数据资料存储、备份、归档等一系列需求。

云存储技术的最大优势是它可以将单一的存储产品转换为数据存储与服务，可以在未来代替传统网络存储并为用户提供更好的服务，而且云存储已经成为未来网络存储技术发展的一个主要趋势。在这种技术下，网络存储才能适应信息社会快速发展的今天。

本章介绍的云文件系统是对开源云计算文件系统 Hadoop HDFS 进行定制与改造，提供面向网

络服务的云计算分布式文件系统。该系统支持采用 Java 编程语言和云文件系统提供的接口实现文件的增加、删除、恢复、上传下载与查询等功能。

系统后台以多机器组成的 Hadoop 集群为基础，前台只需要有 Java 虚拟机和 Tomcat 服务容器即可。后台集群为文件存储和集群操作提供了较为稳定的存储环境，基于 Web 的前台界面为用户提供了方便的操作平台，管理员可以管理普通用户并对集群进行配置，普通用户可以使用基于云端的网盘服务。该系统灵活性较高，同时具有良好的系统性能和用户体验。

本系统具有一定的容错性，适合部署在普通 PC 机或服务器集群上，提供高吞吐量的数据访问，能够支持较大规模的数据存储与访问。

12.2 需求分析与总体设计

12.2.1 需求分析

根据 12.1 节的描述，首先对系统进行需求分析，图 12.1 首先给出了该系统的核心用例图，明确了系统的主要功能需求。

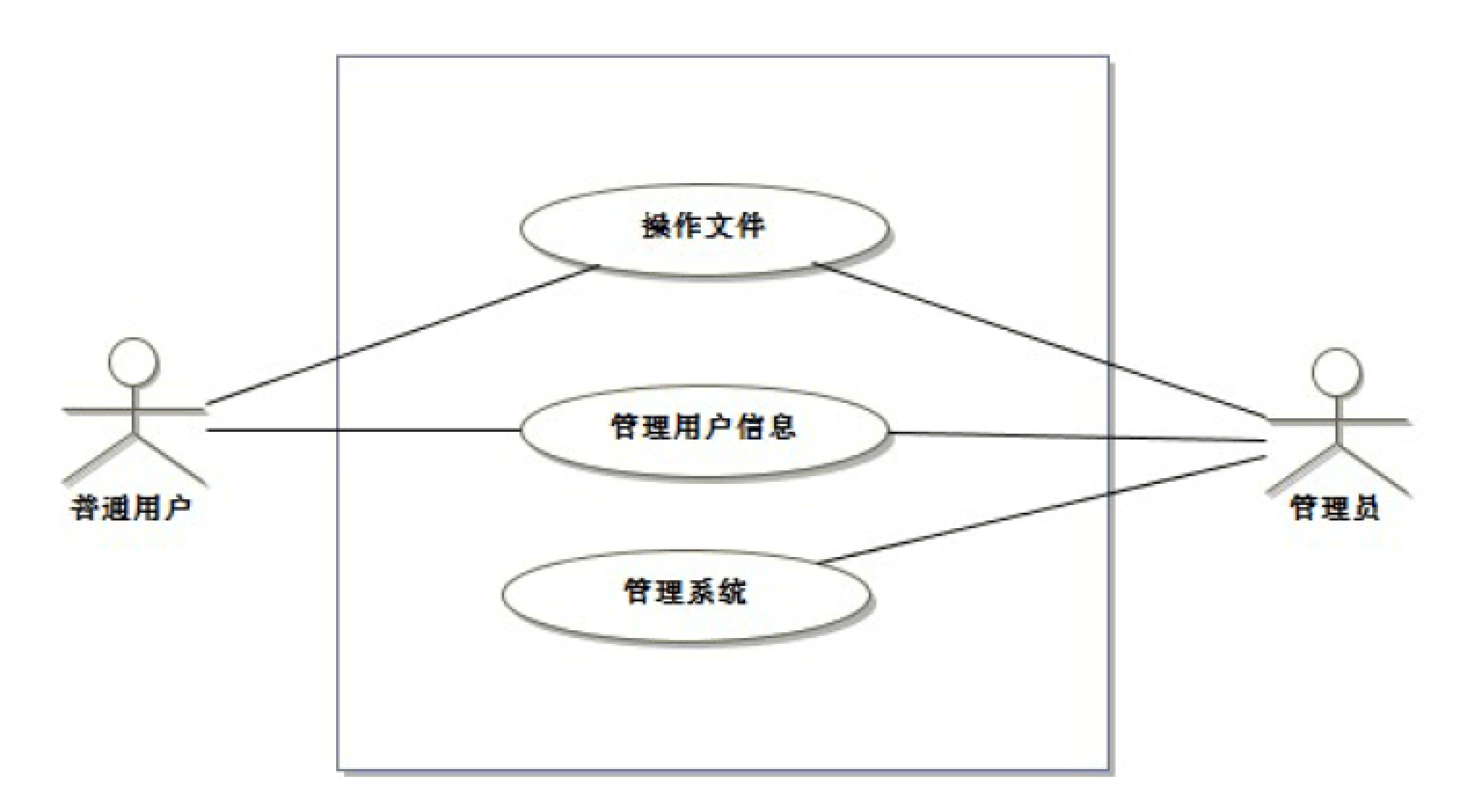


图 12.1　云文件系统核心用例图

从图 12.1 中可以看出，云文件系统有三个核心用例，分别是：普通用户和管理员文件操作，普通用户和管理员进行用户信息管理和管理员进行系统管理。

在核心用例图的基础上，对核心用例进行细化，得到本系统的详细用例，如图 12.2 所示。

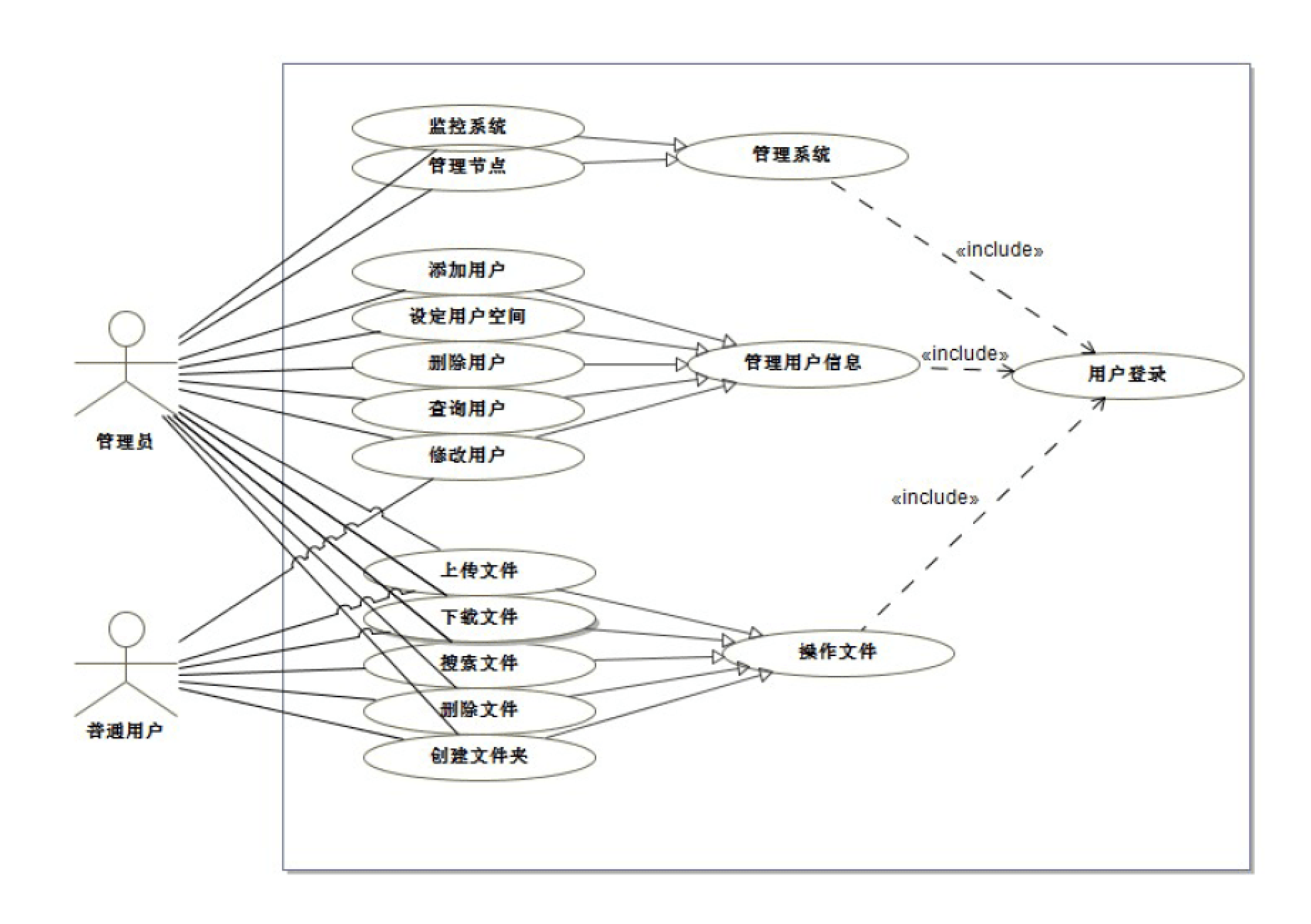


图 12.2　基于 HDFS 的云文件系统详细用例图

从图 12.2 可以看出，普通用户通过登录进入系统后可以管理保存在云文件系统中的自己的文件，还可以通过系统的回收站对自己删除的文件进行恢复、对文件进行彻底删除以及对回收站的清空操作；系统管理员通过登录进入系统后可以对云文件系统后台进行全面的管理操作，主要分为：集群管理，管理员添加或删除用来存储用户文件的后台存储节点；文件管理，管理员查看系统用户的文件；用户管理，管理员添加、删除、修改系统用户信息。

1. 管理系统

管理系统对云文件系统中的服务器集群进行监管和控制，支持集群动态扩展与负载均衡等。系统管理功能细分为系统监控和节点管理两个子功能。

- 监控系统。采用列表、柱状图和饼状图三种方式呈现集群中每个节点的 CPU、内存和存储容量等信息；采用列表方式实时呈现集群中的活动节点、退役中节点和已退役节点。
- 管理节点。输入 IP 段，查找集群中可添加的机器列表，采用可视化的方式向集群中添加新的节点或删除现有节点。

2. 操作文件

对存储在云文件系统中的文件进行上传、下载、删除、搜索和创建文件夹等操作。

- 上传文件。通过 Web 页面选择本地文件，将本地文件上传到云文件系统中，并在云文件系统中保存该文件。
- 下载文件。通过 Web 页面从云文件系统中选择所需文件下载到本地。
- 搜索文件。通过文件名模糊查询云文件系统中的所有文件，并对返回的文件进行下载、重

命名和删除操作。
- 删除文件。文件删除后，首先保存在回收站中，保存期限为 10 天，超过 10 天后系统自动彻底删除回收站中的所有文件；对回收站中的文件执行删除操作后，文件将被彻底删除。
- 创建文件夹。可以在云文件系统中创建和删除文件夹，文件夹可以多层嵌套。

3. 管理用户信息

对使用云文件系统的用户进行管理，用户分为管理员和普通用户两类。普通用户只能操作自己空间内的所有文件与文件夹；管理员除了具有普通用户的所有权限外，还具有系统管理和用户管理两种功能。用户管理细分为添加用户、修改用户、删除用户、查询用户和设定用户空间 5 个子功能。

- 添加用户。填写用户的基本信息和认证信息，明确用户种类，创建该用户。
- 修改用户。对用户基本信息和认证信息进行修改。
- 删除用户。删除用户，包括该用户的认证信息和基本信息，该操作不可恢复。
- 查询用户。采用列表方式显示所有用户的认证信息和基本信息。
- 设定用户空间。在创建用户的过程中设定用户能够拥有的存储空间大小，控制用户的文件存储量。

下面列出了各项功能的详细用例表，如表 12.1~12.15 所示。

表 12.1　管理员登录/退出详细用例表

用例名称	登录/退出
用例描述	管理员通过登录系统进入和退出云文件系统
参与者	管理员和服务器
前置条件	管理员打开登录界面
后置条件	管理员以登录页面显示登录名
基本操作	1. 管理员输入账号密码 2. 提交 3. 中间业务接受输入信息 4. 中间业务查询数据库核对信息 5. 若是信息正确，页面跳转到管理员界面 6. 若信息错误，提示错误信息并重新跳转到登录界面 7. 登录后可以单击“退出”按钮退出系统
业务规则	管理员只能登录一次不能重复登录

从表 12.1 可以看出，管理员登录/退出的主要工作有 7 步，管理员填写账号和密码然后提交，中间业务将信息与数据库中的信息进行核对，如果信息正确则进入系统，如果信息有误系统提示错误信息并跳转到登录界面重新输入。然后开始相应的流程。该用例的主要业务规则是：管理员只能登录一次，不能重复登录，只有正确输入信息后才能进入系统。管理员通过此功能进入系统，

对系统后台服务器和文件系统进行管理和监视。

表 12.2 修改个人信息详细用例表

用例名称	修改个人信息
用例描述	管理员在管理员界面修改自己的信息
参与者	管理员和服务器
前置条件	管理员已经登录
后置条件	数据库中有修改后的信息
基本操作	1. 选择修改信息 2. 填写新信息 3. 提交 4. 中间业务接收输入信息 5. 中间业务修改数据库 6. 返回修改成功
业务规则	管理员可以修改其他所有用户的信息

从表 12.2 可以看出，修改个人信息的主要工作有 6 步，管理员选择用户后单击修改信息，然后开始相应的流程。该用例的其主要业务规则是：管理员可以修改其他所有用户的信息。

表 12.3 集群管理详细用例表

用例名称	集群管理
用例描述	管理员在管理员界面进行云文件系统后台的集群管理
参与者	管理员和服务器
前置条件	管理员已经登录
后置条件	后台有可操作的集群
基本操作	1. 管理员选择操作集群的动作 2. 输入配置信息 3. 中间业务接受选择信息 4. 中间业务按管理员操作处理集群 5. 返回新的集群状态信息
业务规则	管理的集群必须是连接在后台节点上的机器

从表 12.3 可以看出，集群管理的主要工作有 5 步，管理员进入集群管理页面选择相应操作，然后开始相应的流程。该用例的主要业务规则是：管理员管理的集群必须是连接在后台节点上的机器。

表 12.4　文件管理详细用例表

用例名称	文件管理
用例描述	管理员在管理员界面管理系统所有用户的所有文件
参与者	管理员和服务器
前置条件	管理员已经登录
后置条件	文件经过管理
基本操作	1.　管理员选择文件操作类型 2.　中间业务接受操作类型 3.　中间业务遍历文件系统 4.　返回文件操作后的结果
业务规则	管理员只能查看并搜索系统中用户的文件，不能对文件进行增、删、改等内容变更操作

从表 12.4 可以看出，集群管理的主要工作有 4 步，管理员进入文件系统页面选择相应操作，然后开始相应的流程。该用例的主要业务规则是：管理员只能查看并搜索系统中用户的文件，不能对文件进行增、删、改等内容变更操作。

表 12.5　用户管理详细用例表

用例名称	用户管理
用例描述	管理员在管理员界面管理系统用户
参与者	管理员和服务器
前置条件	管理员已经登录
后置条件	系统用户经过更新
基本操作	1.　管理员选择管理用户的类型 2.　填写操作信息 3.　中间业务接受操作类型 4.　中间业务更新数据库 5.　返回更新后的所有用户
业务规则	管理员可以对系统中的用户进行增、删、改、查

从表 12.5 可以看出，用户管理的主要工作有 5 步，管理员进入用户管理页面选择相应操作，然后开始相应的流程。该用例的主要业务规则是：管理员可以对系统中的用户进行增、删、改、查。

表 12.6　用户登录/退出详细用例表

用例名称	登录/退出
用例描述	用户通过登录系统进入和退出云文件系统
参与者	用户和服务器

（续表）

前置条件	用户打开登录界面
后置条件	用户进入用户界面并显示用户登录名
基本操作	1. 用户输入账号密码 2. 中间业务接受输入信息 3. 中间业务查询数据库核对用户信息，如果信息错误跳转到登录界面 4. 登录后可以单击“退出”按钮退出系统
业务规则	用户只能登录一次，不能重复登录

从表 12.6 可以看出，用户登录/退出的主要工作有 4 步，用户进入登录页面填写用户名和密码，然后开始相应的流程。该用例的主要业务规则是：用户只能登录一次，不能重复登录。

表 12.7　修改个人信息详细用例表

用例名称	修改个人信息
用例描述	用户在用户界面修改自己的信息
参与者	用户和服务器
前置条件	用户已经登录
后置条件	数据库中有修改后的用户信息
基本操作	1. 选择修改个人信息 2. 填写新信息 3. 中间业务检查信息的合法性 4. 中间业务接收输入信息 5. 中间业务修改数据库 6. 返回修改成功
业务规则	用户只能修改自己的信息，不能修改管理员给自己分配的空间

从表 12.7 可以看出，用户修改个人信息的主要工作有 6 步，用户进入用户界面选择修改个人信息，然后开始相应的流程。该用例的主要业务规则是：用户只能修改自己的信息，不能修改管理员给自己分配的空间。

表 12.8　搜索文件详细用例表

用例名称	搜索文件
用例描述	用户在用户界面搜索自己的文件
参与者	用户和服务器
前置条件	用户已经登录
后置条件	页面展示按关键字找到的文件

（续表）

基本操作	1.　用户在搜索框填写关键字 2.　提交 3.　中间业务接收数据 4.　中间业务遍历该用户的文件，找到含关键字的文件 5.　返回根据关键字找到的文件
业务规则	用户只能搜索自己的文件

从表 12.8 可以看出，用户搜索文件的主要工作有 5 步，用户进入用户界面在搜索框中填写关键字，然后开始相应的流程。该用例的主要业务规则是：用户只能搜索自己的文件。

表 12.9　上传文件详细用例表

用例名称	上传文件
用例描述	用户在用户界面进入特定目录上传自己的文件
参与者	用户和服务器
前置条件	用户已经登录进入用户界面
后置条件	该目录下有用户上传的文件
基本操作	1.　用户单击上传按钮 2.　页面弹出文件选择框 3.　用户选择文件并确定 4.　中间业务上传该文件 5.　若上传成功更新文件列表 6.　若上传失败显示提示信息并返回文件列表
业务规则	用户上传的文件文件名不能含有特殊字符，目录下不能有同名文件

从表 12.9 可以看出，用户上传文件的主要工作有 6 步，用户进入用户界面单击上传文件按钮，然后开始相应的流程。该用例的主要业务规则是：用户上传的文件文件名不能含有特殊字符，目录下不能有同名文件。

表 12.10　新建目录详细用例表

用例名称	新建目录
用例描述	用户在用户界面进入特定目录新建文件夹
参与者	用户和服务器
前置条件	用户已经登录进入用户界面
后置条件	该目录下有用户刚才新建的文件夹
基本操作	1.　用户单击新建文件夹按钮 2.　页面弹出文件名输入框 3.　用户输入文件名并提交 4.　中间业务创建该文件夹 5.　若创建成功返回含有新建文件夹的文件列表，否则显示错误信息并返回文件列表
业务规则	输入的文件名不能含有特殊字符，不能同所在目录下的文件重名

从表 12.10 可以看出，用户新建文件夹的主要工作有 5 步，用户进入用户界面单击新建文件

夹按钮，然后开始相应的流程。该用例的主要业务规则是：输入的文件名不能含有特殊字符，不能同所在目录下的文件重名。

表 12.11　回收站管理详细用例表

用例名称	回收站管理
用例描述	用户进入回收站页面管理自己删掉的文件
参与者	用户和服务器
前置条件	用户已经登录进入回收站页面
后置条件	回收站内容已经更新
基本操作	1. 用户选择回收站中文件操作类型 2. 中间业务判断文件操作类型 3. 中间业务操作回收站中的文件，若是彻底删除将文件从回收站删除，若是恢复文件将文件从回收站恢复，若是清空回收站则删除回收站中所有的文件 4. 返回用户管理后的回收站文件列表
业务规则	从回收站删除或清空文件后文件不可恢复

从表 12.11 可以看出，用户回收站管理的主要工作有 4 步，用户进入回收站界面选择操作，然后开始相应的流程。该用例的主要业务规则是：从回收站删除或清空文件后文件不可恢复。

表 12.12　文件重命名详细用例表

用例名称	文件重命名
用例描述	用户修改文件名称
参与者	用户和服务器
前置条件	用户已经登录进入用户页面
后置条件	文件名被修改
基本操作	1. 用户选择文件后的重命名按钮 2. 系统跳出填写文件名的输入框 3. 用户输入文件名 4. 中间业务接受输入信息并修改文件名 5. 若修改成功返回修改后的文件列表，否则显示错误信息然后返回修改前的文件列表
业务规则	修改的文件名不能含有特殊字符，不能和同目录下的文件名相同

从表 12.12 可以看出，用户重命名文件的主要工作有 5 步，用户进入用户界面单击文件后面的重命名按钮，然后开始相应的流程。该用例的其主要业务规则是：修改的文件名不能含有特殊字符，不能和同目录下的文件名相同。

表 12.13　文件下载详细用例表

用例名称	文件下载
用例描述	用户下载自己的文件
参与者	用户和服务器
前置条件	用户已经登录进入用户页面
后置条件	文件被下载到用户电脑
基本操作	1.　用户选择文件后的下载按钮 2.　页面弹出保存路径选择框 3.　用户选择路径 4.　中间业务执行文件下载 5.　下载完成后返回下载前的页面
业务规则	只能下载文件而不能下载文件夹

从表 12.13 可以看出，用户文件下载的主要工作有 5 步，用户进入用户界面单击文件后面的下载按钮，然后开始相应的流程。该用例的主要业务规则是：只能下载文件而不能下载文件夹。

表 12.14　文件删除详细用例表

用例名称	文件删除
用例描述	用户删除目录中的文件
参与者	用户和服务器
前置条件	用户已经登录进入用户页面
后置条件	文件删除
基本操作	1.　用户选择文件后的删除按钮 2.　页面弹出是否删除选择框 3.　用户选择确定/取消 4.　中间业务层接受信息，如果是“确定”执行删除文件操作将文件删除到回收站中并返回，如果是“取消”则取消操作并返回
业务规则	文件被删除到回收站，可以从回收站恢复

从表 12.14 可以看出，用户文件删除的主要工作有 4 步，用户进入用户界面单击文件后面的删除按钮，然后开始相应的流程。该用例的主要业务规则是：文件被删除到回收站，可以从回收站恢复。

表 12.15　文件分类检索详细用例表

用例名称	文件分类检索
用例描述	用户按类别查找自己的文件
参与者	用户和服务器

（续表）

前置条件	用户已经登录进入用户页面
后置条件	显示用户所选择的类别的所有文件
基本操作	1. 用户选择用户页面左边的类别选项 2. 中间业务接受类别信息 3. 中间业务按类别遍历用户文件，找出该类别的文件并以列表的形式显示给用户
业务规则	按类别检索只是对文件的检索

从表 12.15 可以看出，用户文件分类检索的主要工作有 3 步，用户进入用户界面单击页面左边的类别选项，然后开始相应的流程。该用例的主要业务规则是：按类别检索只是对文件的检索。

系统数据的来源方式主要有两种，即管理员对系统的管理和用户的文件操作，具体系统上下文数据流图如图 12.3 所示。

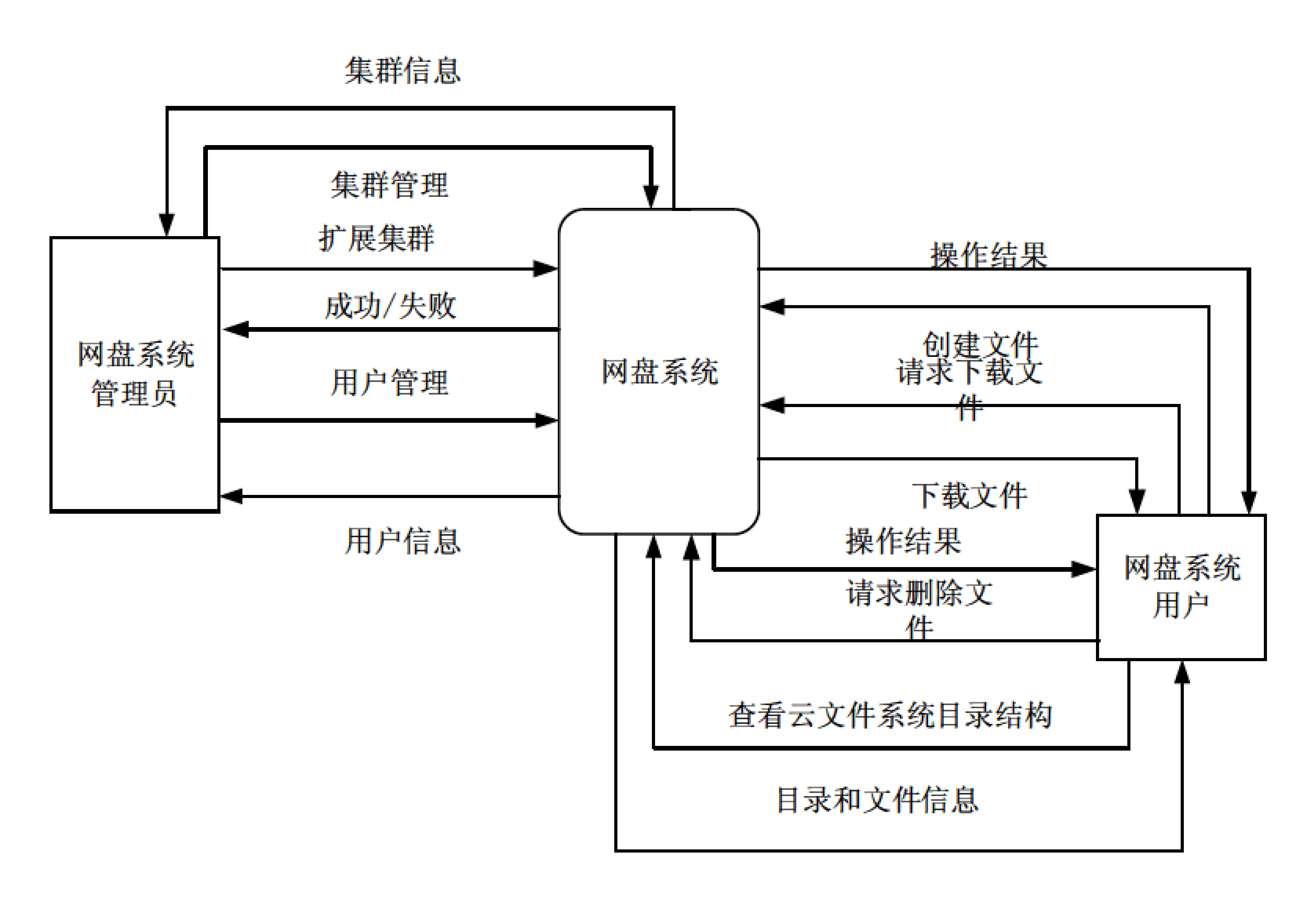


图 12.3　基于 HDFS 的云文件系统的上下文数据流图

从图 12.3 可以看出，云文件系统会与系统管理员、系统用户等外部代理产生一定的数据读写工作，管理员通过管理系统后台与系统进行交互，系统用户通过管理自己的文件与系统进行交互并明确给出了系统的边界。

12.2.2　总体设计

1. 系统功能设计

通过对系统的需求分析，可以看出系统主要分为三大功能模块，具体如图 12.4 所示。

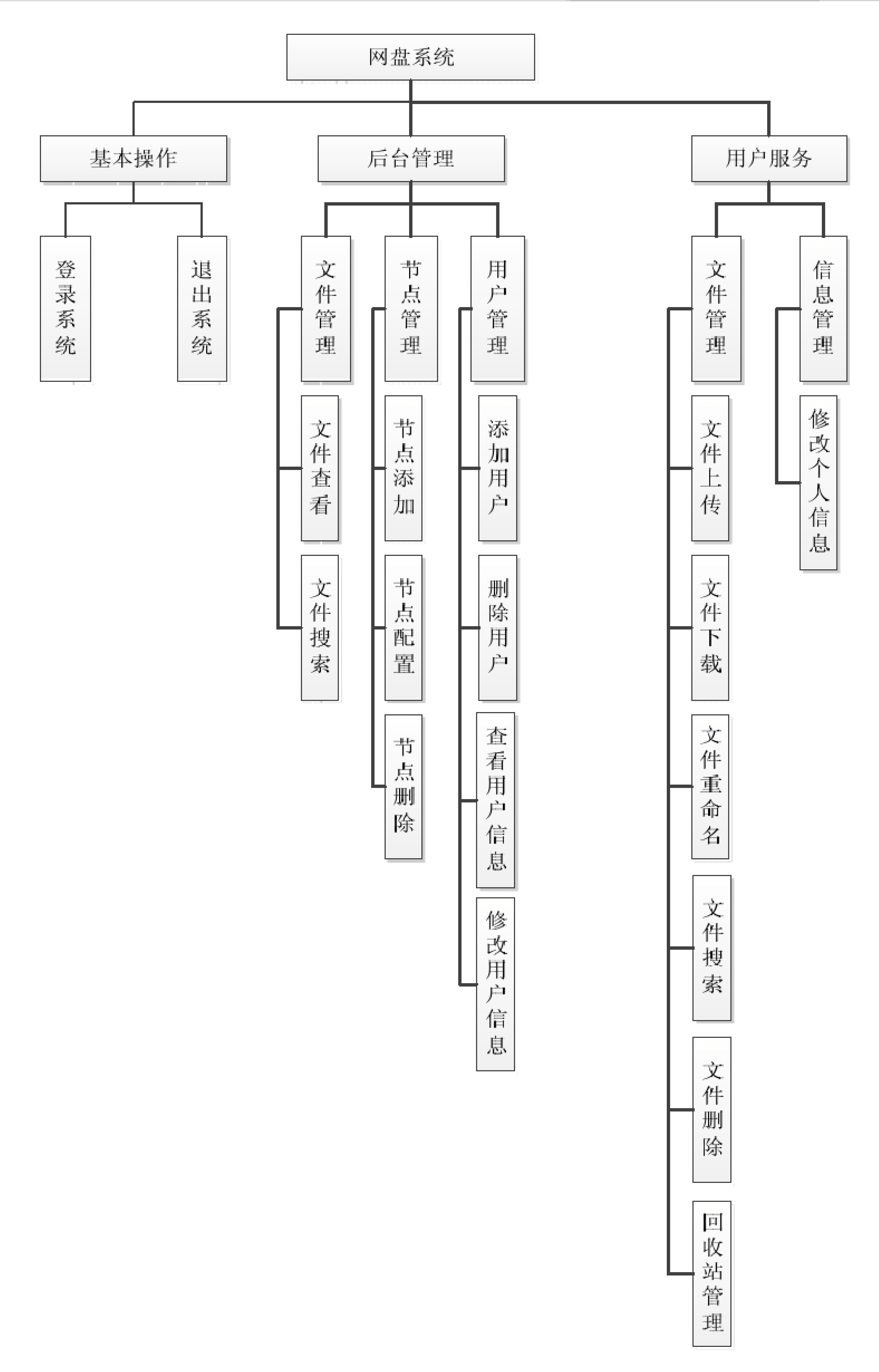


图 12.4　基于 HDFS 的云文件系统功能分解图

从图 12.4 可以看出，云文件系统分为基本操作、后台管理、用户服务三个模块，具体功能如下。

（1）基本操作模块

- 登录系统：基于用户名和密码的方式，提供给管理员和用户进入本系统的方法。验证成功后，管理员进入管理界面，用户进入用户界面。
- 退出登录：提供给已进入本系统的管理员和用户退出自己账户的功能。

（2）后台管理模块

- 文件管理：文件管理是提供给系统管理员的功能，管理员可以查看和搜索系统中所有用户的文件。
- 节点管理：节点管理是提供给系统管理员的功能，管理员可以向系统添加节点、删除节点、对系统中的节点进行配置。
- 用户管理：用户管理是提供给系统管理员的功能，管理员可以添加用户、删除用户、查看用户信息、修改用户信息、给用户分配空间等。

（3）用户服务模块

- 文件管理：文件管理是提供给系统用户的功能，用户可以向本系统自己的空间中上传文件、创建文件夹、下载文件、重命名文件、搜索文件、删除文件以及在回收站里恢复文件，还可以清空回收站。
- 修改个人信息：修改个人信息是提供给系统用户的功能，用户可以修改自己的信息。

针对系统的功能需求，设计了系统的架构为 B/S 架构。B/S 架构具有分布性特点，可以随时随地进行查询、浏览等业务处理；业务扩展简单方便，通过增加页面即可增加服务器功能；维护简单方便，只需要改变网面，即可实现所有用户的同步更新。因此选择 B/S 作为本系统的基本架构，系统集群架构图如图 12.5 所示。

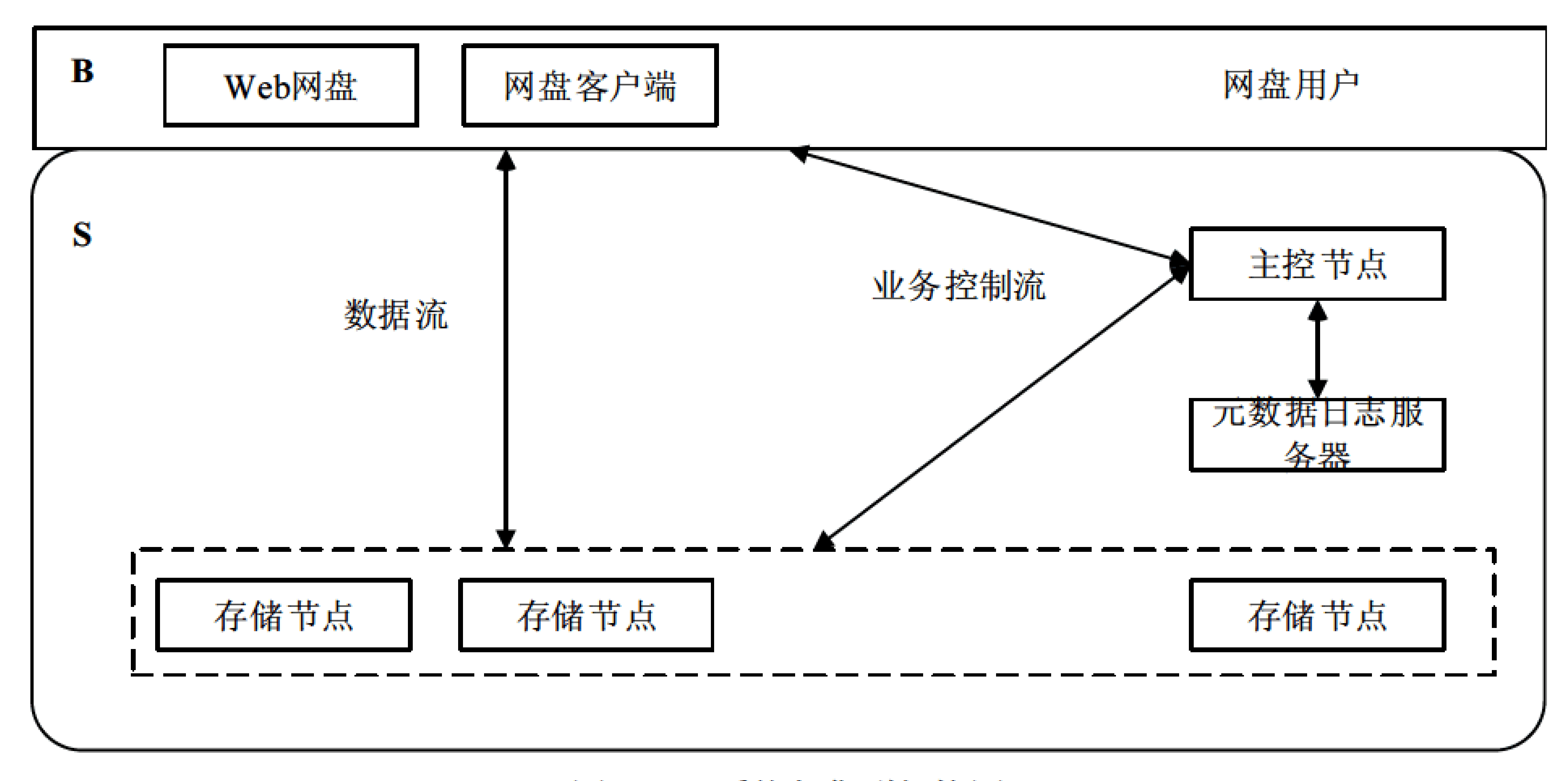


图 12.5　系统与集群架构图

2. 数据备份方案设计

（1）Hadoop 的元数据备份方案

该方案利用 Hadoop 自身的 Failover 措施（通过配置 dfs.name.dir），NameNode 可以将元数据信息保存到多个目录。通常的做法是，选择一个本地目录、一个远程目录（通过 NFS 进行共享），当 NameNode 发生故障时，可以启动备用机器的 NameNode，加载远程目录中的元数据信息，提供服务。

（2）Hadoop 的 Secondary NameNode 方案

该方案启动一个 Secondary NameNode 节点，该节点定期从 NameNode 节点上下载元数据信息（元数据镜像 fsimage 和元数据库操作日志 edits），然后将 fsimage 和 edits 进行合并，生成新的 fsimage（该 fsimage 就是 Secondary NameNode 下载时刻的元数据的 Checkpoint），在本地保存，并将其推送到 NameNode，同时重置 NameNode 上的 edits。

（3）Hadoop 的 Checkpoint Node 方案

Checkpoint Node 方案与 Secondary NameNode 的原理基本相同，只是实现方式不同。该方案利用 Hadoop 的 Checkpoint 机制进行备份，配置一个 Checkpoint Node。该节点会定期从 Primary NameNode 中下载元数据信息（fsimage+edits），将 edits 与 fsimage 进行合并，在本地形成最新的 Checkpoint，并上传到 Primary NameNode 进行更新。

当 NameNode 发生故障时，极端情况下（NameNode 彻底无法恢复），可以在备用节点上启动一个 NameNode，读取 Checkpoint 信息，提供服务。

（4）Hadoop 的 Backup Node 方案

利用新版本 Hadoop 自身的 Failover 措施，配置一个 Backup Node，Backup Node 在内存和本地磁盘均保存了 HDFS 系统最新的名字空间元数据信息。如果 NameNode 发生故障，可使用 Backup Node 中最新的元数据信息。

12.3 相关技术介绍

在上一节我们对云文件系统有了基本的认识，本节将介绍本云文件系统的开发细节，详细介绍本系统涉及的集中核心技术和理论知识。

系统的开发平台采用的是 MyEclipse + Tomcat + MySQL+ Hadoop，下面将分别对本系统所用到的技术和主要开发工具进行介绍。

12.3.1 Hadoop HDFS 介绍

Hadoop 是由 Apache Lucene 创始人 Doug Cutting 创建的一个分布式系统基础架构。它起源于一个开源的搜索引擎 Apache Nutch，其本身也是 Lucene 项目的一部分。Hadoop 充分利用集群高速运算和存储的威力实现了一个分布式的文件系统（Hadoop Distributed File System，HDFS）。HDFS 有高容错性的特点，可以用来部署在低廉的硬件上。而且它提供高传输率来访问应用程序的数据，适合那些有着超大数据集的应用程序。HDFS 放宽了对 POSIX 的要求，可支持以流的形式访问文件系统中的数据。

Hadoop 由多个部分构成，其最底层是 Hadoop Distributed File System（HDFS），它存储着 Hadoop 集群中所有存储节点上的文件。HDFS 的上一层是由 JobTrackers 和 TaskTrackers 组成的

MapReduce 引擎。Hadoop 架构如图 12.6 所示。

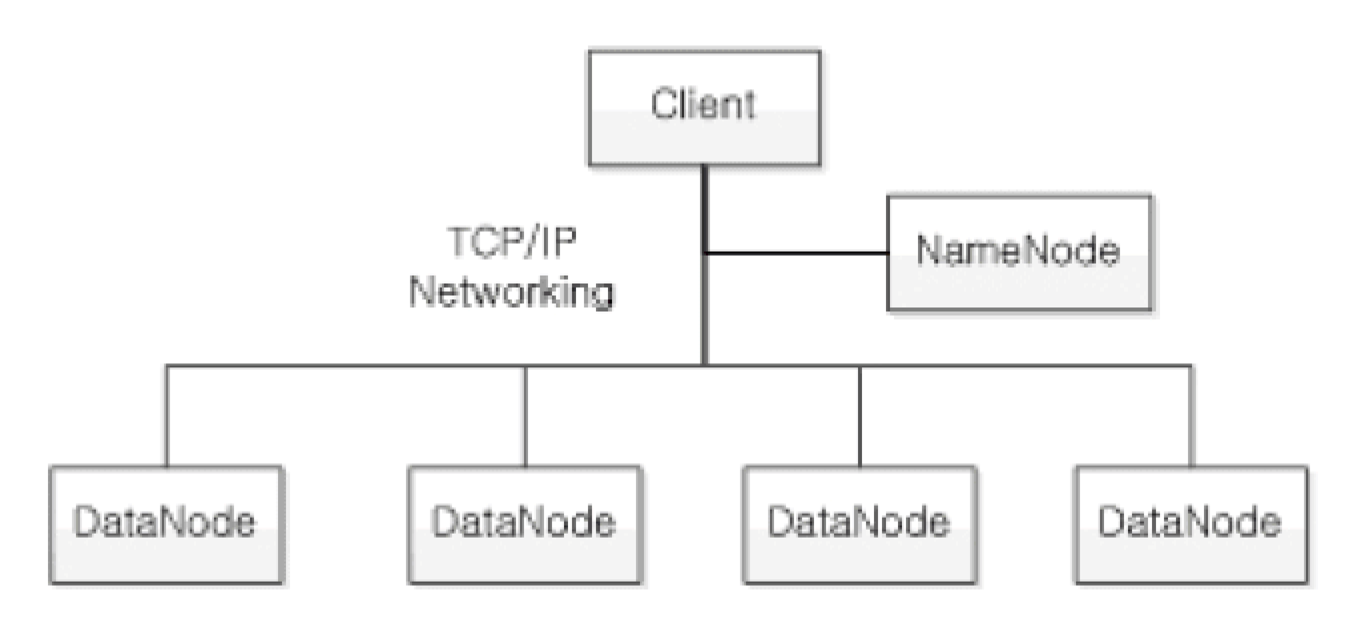


图 12.6　Hadoop 集群的简化视图

如图 12.6 所示，Hadoop 简单地由三部分组成：客户端、NameNode 节点和后台集群，客户端通过 TCP/IP 协议与其他部分进行通信。

HDFS 是一个典型的主/从架构，它包括一个 NameNode 节点（主节点）和多个 DataNode 节点（从节点）并提供应用程序的访问接口。NameNode 是整个文件系统的管理节点，它负责文件系统名字空间的管理与维护，同时负责客户端文件的操作的控制以及集体存储的管理与分配；DataNode 提供真实文件数据的存储服务。对普通客户而言，HDFS 就是一个传统的分级文件系统，可以创建、删除、移动或重命名文件等。HDFS 的一个仅存的缺点是只有一个 NameNode。

文件在 HDFS 中被分成块，然后主机将这些块复制到多个计算机（DataNode）中。块的大小通常为 64MB，在创建文件时由客户机决定复制的块数量。NameNode 可以控制所有文件操作。HDFS 内是基于标准的 TCP/IP 协议进行通信的。

NameNode 在 HDFS 中单独机器上运行，负责控制外部客户端的访问并管理文件系统的名称空间。NameNode 决定了文件在聚群上面的分布情况。常见的有 3 个复制块，第一个复制块存储在同一个机架的不同节点上，最后一个复制块存储在不同机架的不同节点上。实际的 I/O 操作没有经过 NameNode，经过 NameNode 的数据只有表示 DataNode 和块的文件映射的元数据。当外部客户机发送要求创建文件的请求时，NameNode 会以该块的第一个副本的 DataNode IP 地址和块标识作为响应。该 NameNode 还会通知其他将要接收该块的副本的 DataNode。

NameNode 在一个称为 FsImage 的文件中存储所有关于文件系统名称空间的信息。这个文件和一个包含所有事务的记录文件（这里是 EditLog）将存储在 NameNode 的本地文件系统上。FsImage 和 EditLog 文件也需要复制副本，以防文件损坏或 NameNode 系统丢失。

DataNode 通常在 HDFS 实例中的单独机器上运行。Hadoop 集群包含一个 NameNode 和大量的 DataNode，DataNode 的结构普遍是由机架组成的，机架通过一个交换机来连接所有的系统。

DataNode 响应来自 HDFS 客户端的读写操作和来自 NameNode 的对数据的创建、删除和块的复制命令。NameNode 处理响应依靠每个 DataNode 的定期心跳消息的传递。每条心跳消息都包含了一个块信息报告，NameNode 根据这个报告来验证块的映射和其他文件系统的元数据。若 DataNode 不能正常发送心跳消息，NameNode 将进行修复，该操作将重新复制在该节点上所有丢失的块。

HDFS 的整体架构如图 12.7 所示，整个系统可以分为三个部分，即客户端、主控节点和数据

节点，其设计目标是提供一个云文件系统。系统将目录结构和文件内容通过网络存储在远端系统中，而不是集中存储在本地磁盘中。

主控节点是 HDFS 的管理者，主要负责文件系统的命名空间、集群的配置信息和数据块的复制信息等，并将文件系统的元数据存储在内存中；数据节点是文件实际存储的位置，它将数据块信息存储在本地文件系统中，并且通过周期性的心跳报文将所有数据块信息发送给主控节点。系统整体结构如图 12.7 所示。

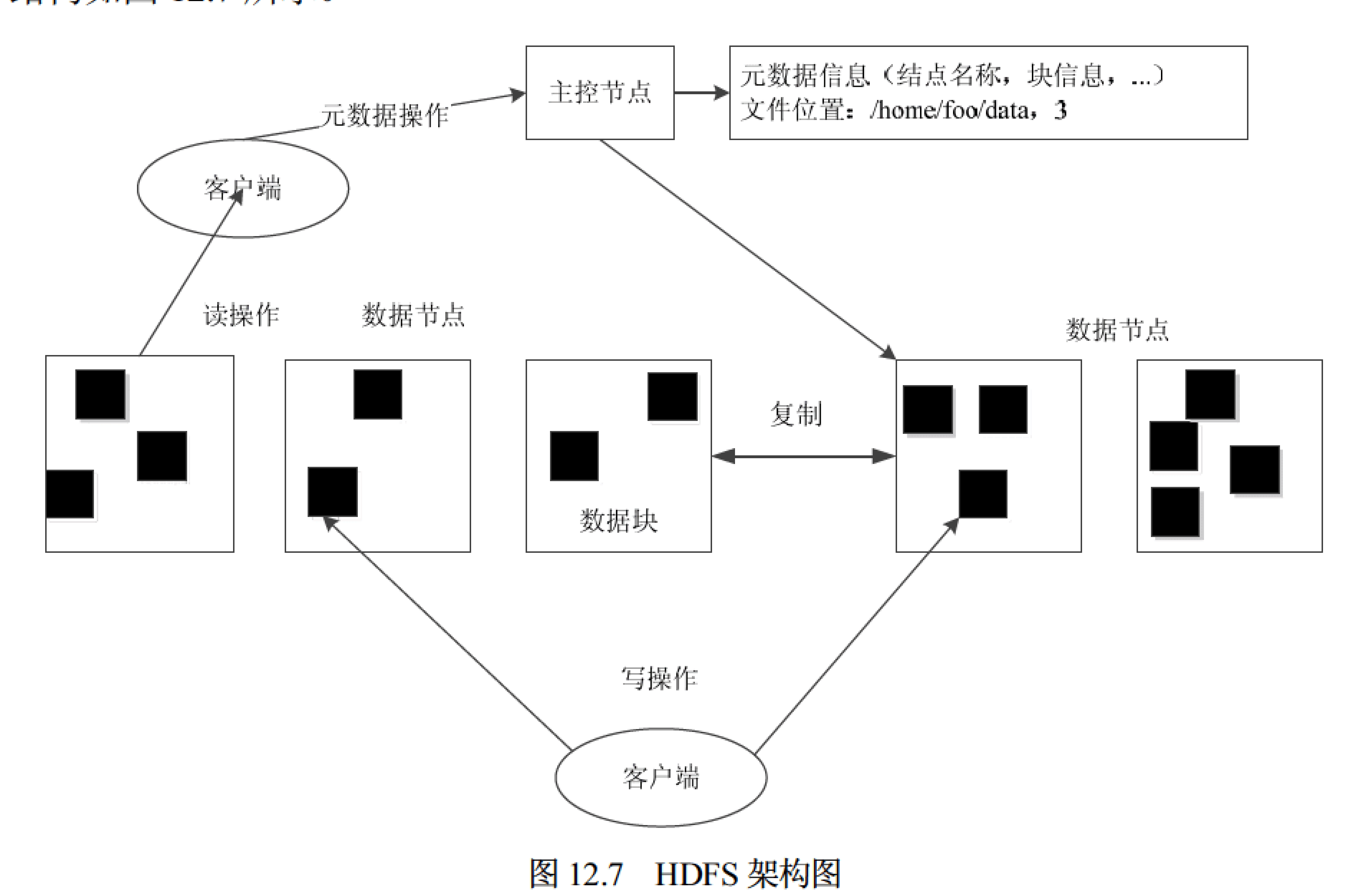


图 12.7　HDFS 架构图

12.3.2　主控节点和数据节点

HDFS 采用的是主/从架构。HDFS 集群是由一个主控节点和一些数据节点组成的。主控节点是一个中心服务器，它负责管理 HDFS 云文件系统的名字空间（namespace）和客户端对文件的访问操作。集群中的数据节点一般是一个管理点，负责管理它所在节点上的存储情况。

HDFS 显示了文件系统的名字空间，用户可以以文件的形式存储自己的数据。在文件系统内部，一个完整的文件其实被分成了一个或多个数据块，这些数据块存储在一组数据节点上。主控节点负责管理文件系统的名字空间，如打开文件、关闭文件、重命名文件或目录等。它也用来确定数据块到具体数据节点的映射方式。文件系统客户端的读写请求由数据节点负责处理。对数据块的创建、删除和复制等操作也是在主控节点的统一调度下进行的。HDFS 默认的数据块大小为 64MB，副本因子为 3。整个集群中，能够同时提供服务的只有主控节点。每个数据节点周期性地发送心跳报文给主控节点，心跳报文发送的成功与否决定了该数据节点能否正常工作。

主控节点和数据节点可以在普通的商用机上运行，这些机器通常运行着 GNU/Linux 操作系统。HDFS 采用的是 Java 语言开发。主控节点或数据节点可以部署在任何支持 Java 的机器上。由于 Java 语言的可移植性极强，HDFS 可以部署到各种类型的机器上。其中一个典型的部署实例是

一台机器上只运行一个主控节点，但是集群中的其他机器分别运行一个数据节点。这种架构并不排斥在一台机器上运行多个数据节点。集群中单一主控节点的结构在很大程度上简化了系统架构。主控节点是所有 HDFS 元数据的管理者和仲裁者，它也是 HDFS 的核心模块。

在 HDFS 中， HDFS 云文件系统的元数据存储在主控节点上。主控节点中的逻辑相对于数据节点而言更复杂但是数据量并不大。主控节点将文件数据被拆分成为多个数据块存储在不同的数据节点中。每个数据块在数据节点中都为一对文件：一个是数据文件，另一个是含有附加信息的元数据文件。

所有的 HDFS 内部通信都是构建在 TCP/IP 协议上的。HDFS 通过一套 RPC 机制来实现各服务间通信的协议。每一对服务器间的通信协议，都被定义为在独立的线程中处理 RPC 请求的一个接口。

12.3.3 页面展现技术

HTML（Hypertext Markup Language）是一种规范，一种标准，它通过标记符号来标记要显示的网页中内容的各个部分。网页文件本身是一种文本文件，通过在文本文件中添加标记符，可以告诉浏览器如何显示其中的内容，浏览器按顺序来解析网页文件，然后根据标记符解释和显示其标记的内容，它将忽略网页文件中书写出错的标记，且不停止其解释执行过程，通过显示效果，编程者可以分析出错原因和出错位置。但对于不同的浏览器，对同一标签可能会有不完全相同的解释，而会出现不同的显示效果。

JavaScript 是一种基于对象和事件驱动并具有相对安全性的客户端脚本语言。它可以给 HTML 网页添加动态功能，如响应用户的各种操作。它是一种弱类型、动态、基于原型的语言，可以支持类。JavaScript 也可以用于服务器端编程等其他场合。完整的 JavaScript 包含三个部分：字节顺序记号，文档对象模型，ECMAScript。

CSS（Cascading Style Sheet）可译为“层叠样式表”，通常用它来定义如何显示 HTML 元素，它用于控制 Web 页面的外观。通过使用 CSS 实现页面的内容与表现形式分离。

12.3.4 页面控制技术

Servlet 是在服务器上运行的小程序。这个词是在 Java Applet 的环境中创造的，Java Applet 是一种当作单独文件跟网页一起发送的小程序，它通常在服务器端运行，为用户进行运算或者根据用户互作用定位图形等。

服务器上需要一些程序，常常是根据用户输入访问数据库的程序。这些通常是使用公共网关接口（Common Gateway Interface，CGI）应用程序完成的。然而，在服务器上运行 Java，这种程序可使用 Java 编程语言实现。在通信量大的服务器上，Java Servlet 的优点在于它们的执行速度快于 CGI 程序。各个用户请求被激活成单个程序中的一个线程，而无需创建单独的进程，这意味着服务器端处理请求的系统开销将明显降低。

JSP 全名为 Java Server Page，它是由 Sun Microsystems 公司倡导、许多公司参与建立的一种

动态技术标准。JSP 网页就是在 HTML 文件中加入 Java 程序片段和 JSP 标签。其中 Java 程序片段可以操纵数据库、执行网页的重新定向以及发送 E-mail 等。在服务器端执行了所有的程序操作，它仅仅将结果通过网络传送给客户端，这样大大降低了对客户端配置的要求，即使客户浏览器端不支持 Java，也可以访问 JSP 网页。

JSP 是一个简化的 Servlet 设计，它完成了 HTML 语法对 Java 的扩张。与 Servlet 一样，JSP 运行在服务器端，它送回给客户端浏览器的只是一个 HTML 文本，客户端有浏览器就能浏览内容。收到 JSP 网页的请求时 Web 服务器首先执行请求的程序段，然后将执行结果以 HTML 代码的形式返回给客户端浏览器。插入的 Java 程序段可以操作数据库并对网页进行重新定向等操作，它实现了动态网页所需要的所有基本功能。

JSP 技术使用 Java 编程语言编写类 XML 的 tags 和 scriptlets，来封装产生动态网页的处理逻辑。网页还能通过 tags 和 scriptlets 访问存在于服务端的资源的应用逻辑。JSP 将网页逻辑与网页设计的显示分离，支持可重用的基于组件的设计，使基于 Web 的应用程序的开发变得迅速和容易。JSP（Java Server Pages）是一种动态页面技术，它的主要目的是将表示逻辑从 Servlet 中分离出来。

JSP 页面是由 HTML 代码和 Java 代码组成的。服务器得到客户端的请求后处理这些 Java 代码，然后服务器将生成的HTML 页面结果返回给客户端。JSP 的技术基础是 Java Servlet，Java Servlet 和 JSP 配合可以完成大型的 Web 应用程序的开发需要。JSP 具有完全的面向对象，简单易用，平台无关性，安全可靠，面向因特网等特点。

12.4 详细设计与实现

前面介绍了本系统的具体需求，本节将对本系统设计与实现进行具体的介绍，并将给出部分功能模块的关键代码。

12.4.1　云文件系统的操作流程

通过系统的结构分析，设计程序的流程图如图 12.8 所示。

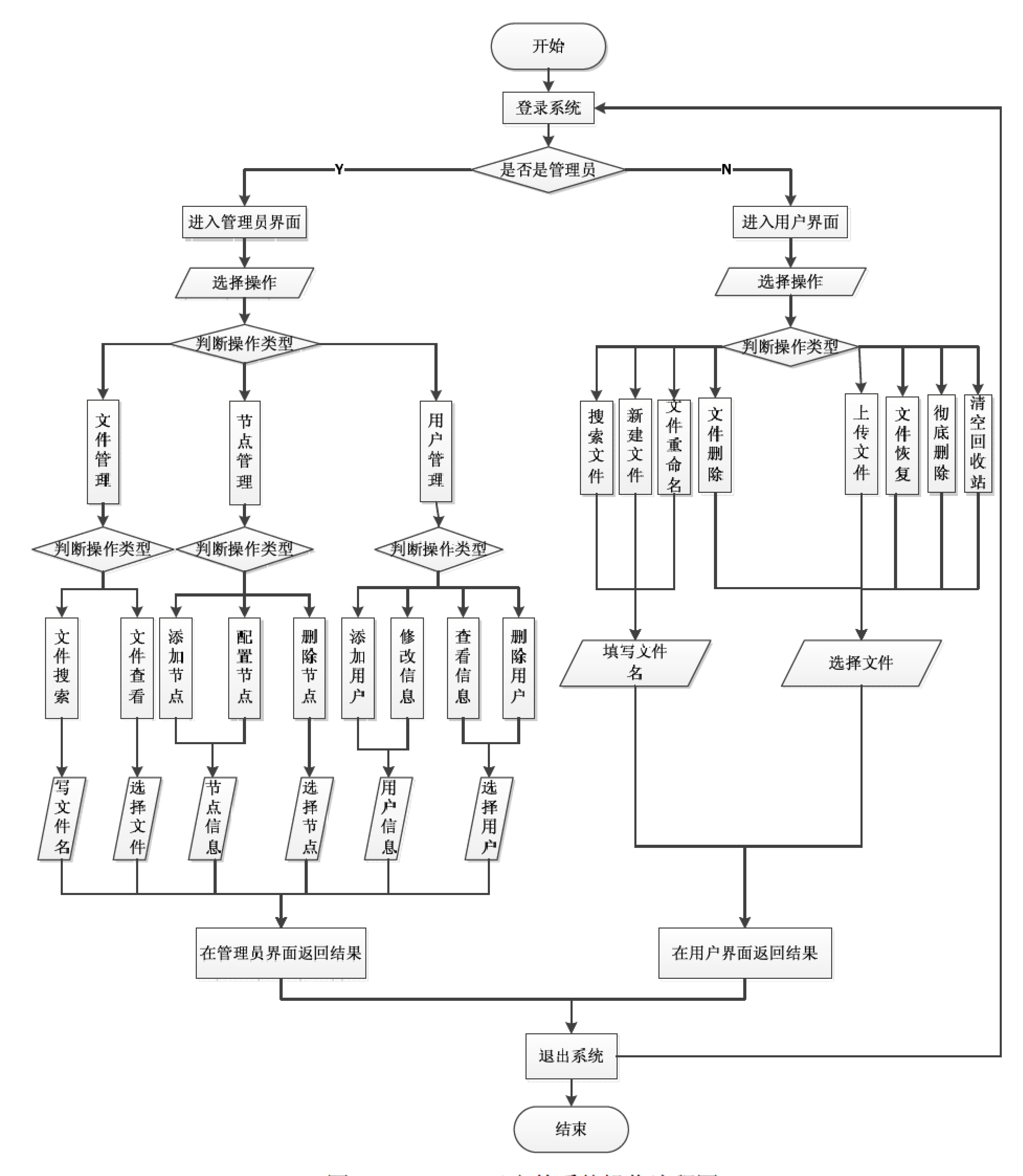


图 12.8　HDFS 云文件系统操作流程图

从图 12.8 可以看出，管理员和用户通过登录进入系统，操作管理系统或管理自己的文件，图示明确给出了系统详细的操作流程。

12.4.2　云文件系统的模块设计

云文件系统的功能模块主要由基本操作模块、后台管理模块、用户服务模块组成。

1. 基本操作模块

在基本操作模块中的登录模块最主要的是身份验证功能，使用本系统之前必须首先通过身份验证。其目的在于维护系统安全性并对用户类别进行区分，管理员将进入管理员界面，用户进入

用户界面。其功能在于对申请登录用户进行身份验证，验证通过者才可以进入系统。用户登录模块的 IPO 图表示如图 12.9 所示。

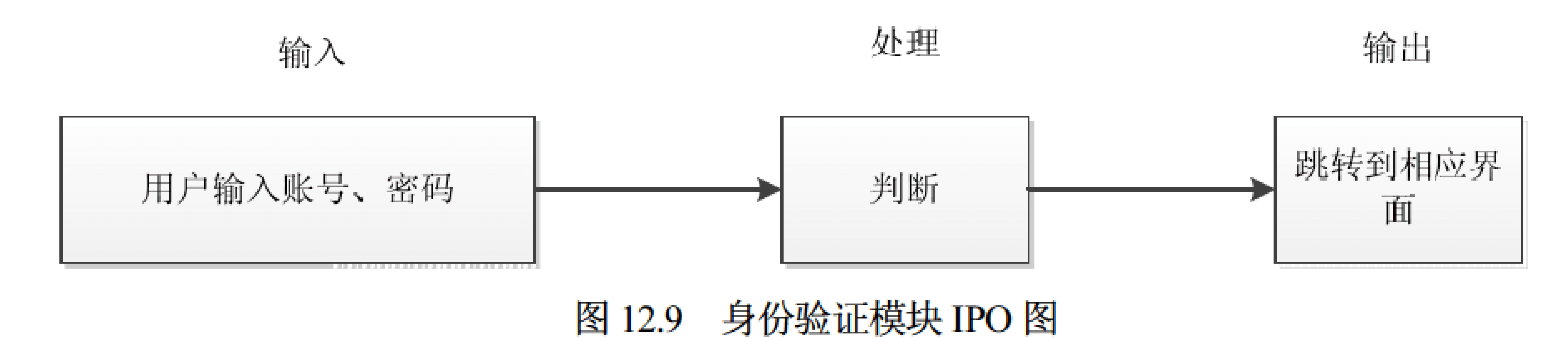


图 12.9　身份验证模块 IPO 图

从图 12.9 可以看出用户输入账号和密码后，登录模块将信息和数据库中的信息进行核对，若正确则跳转到用户界面，若信息错误则提示错误信息并跳转到登录界面。

本模块的逻辑流程用时序图表示如图 12.10 所示。

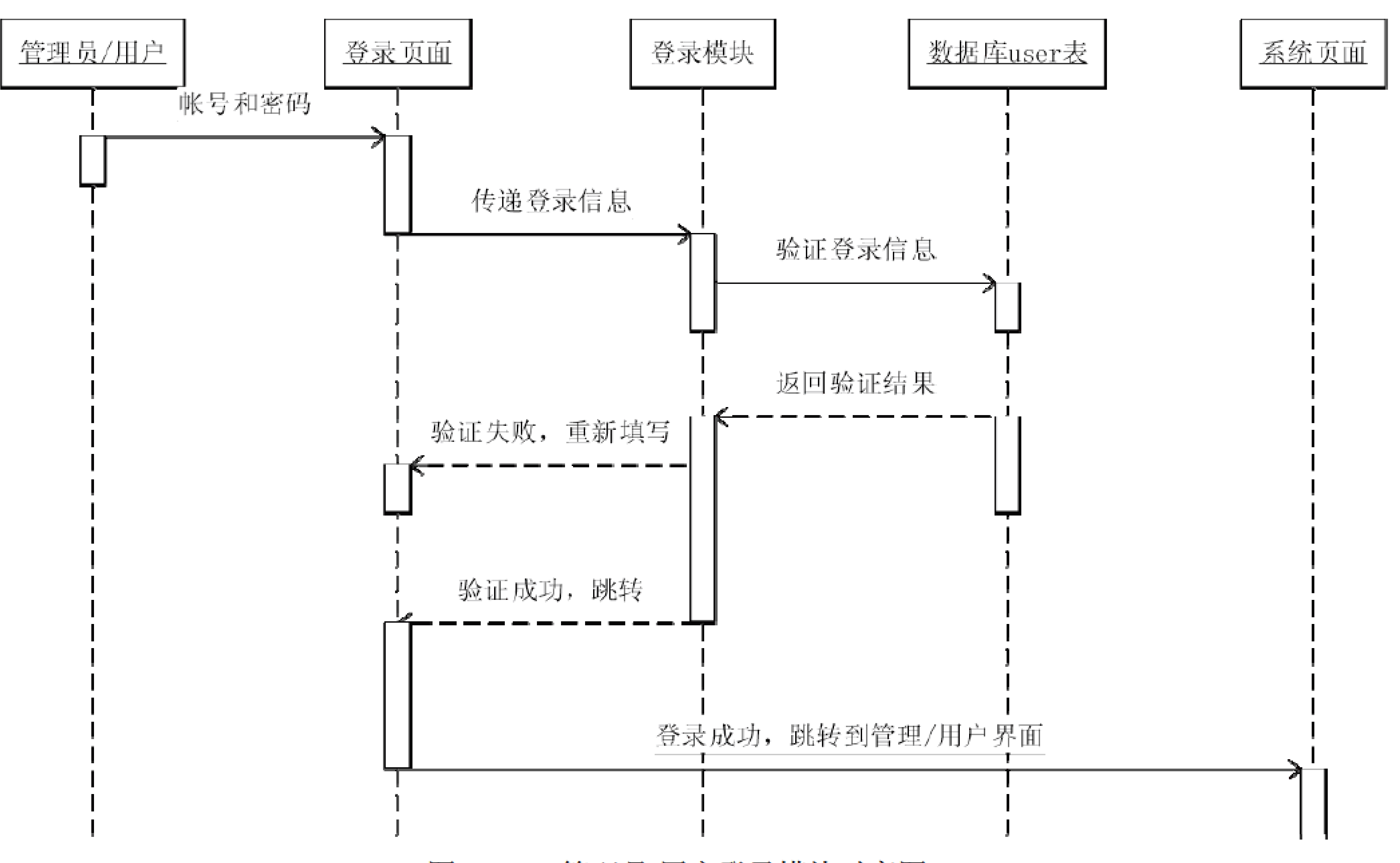


图 12.10　管理员/用户登录模块时序图

从图 12.10 可以看出，管理员或用户在登录页面输入用户名和密码后单击“登录”按钮，系统会调用登录模块中的 login 方法对输入的信息进行处理，将输入的信息与数据库 user 表中信息作对比并验证其权限，完全符合后返回登录成功标志并跳转到对应的系统主页（管理员进入管理界面，用户进入用户界面），若信息不符，则返回错误提示信息并重新跳转到登录界面。

2. 后台管理模块

后台管理支持管理员对系统的文件查看、用户管理、集群管理，具体实现如下。

（1）文件管理模块设计

系统的管理员具有管理系统所有文件的功能，此功能赋予管理员查看并搜索系统中所有用户文件的权限，以便查看系统的运作情况等信息。文件管理的 IPO 图表示如图 12.11 所示。

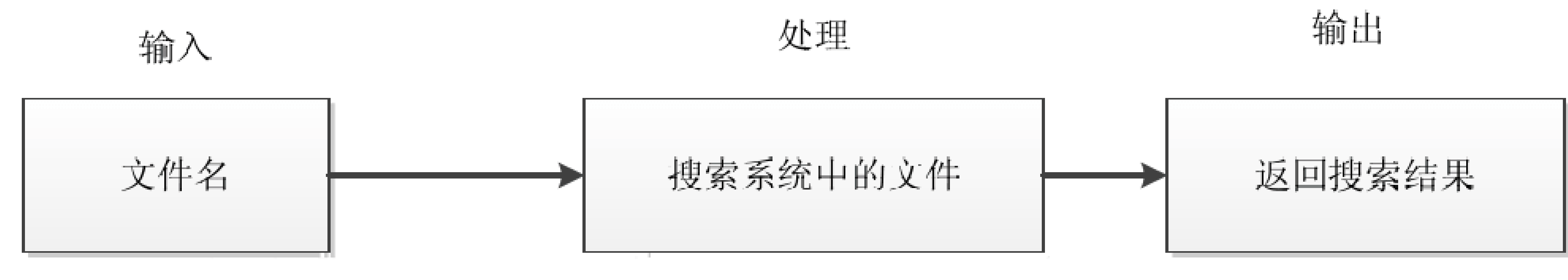


图 12.11　文件管理模块 IPO 图

从图 12.11 可以看出用户输入文件名，文件操作模块根据文件名进行检索并返回检索结果。

本模块的逻辑流程用时序图表示如图 12.12 所示。

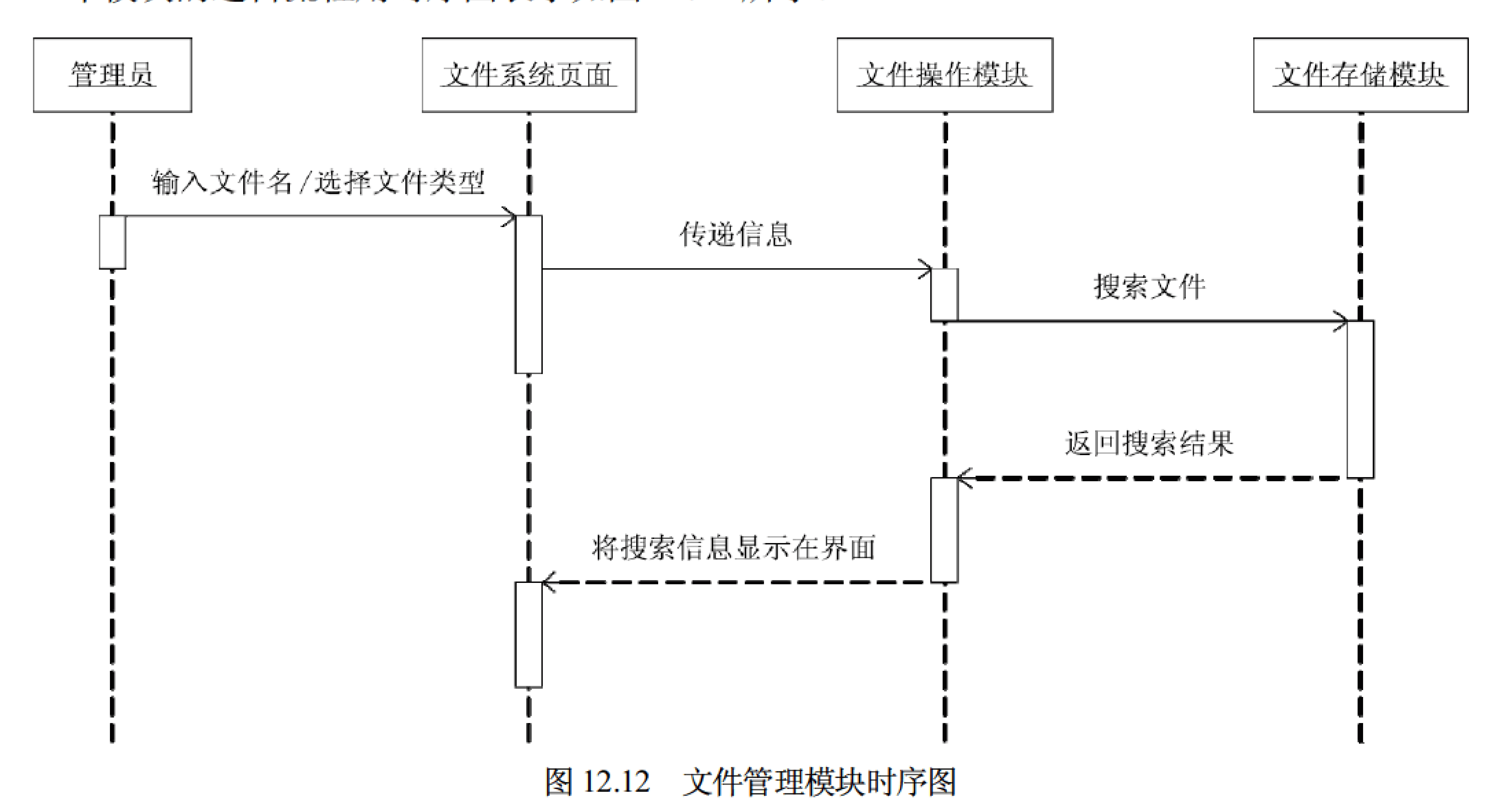


图 12.12　文件管理模块时序图

从图 12.12 可以看出，管理员可以通过输入文件名或选择文件类型搜索系统中的所有文件，图中给出了明确的操作过程。

（2）节点管理模块设计

系统的节点管理模块赋予管理员管理系统后台集群的功能。节点管理模块的 IPO 图表示如图 12.13 所示。

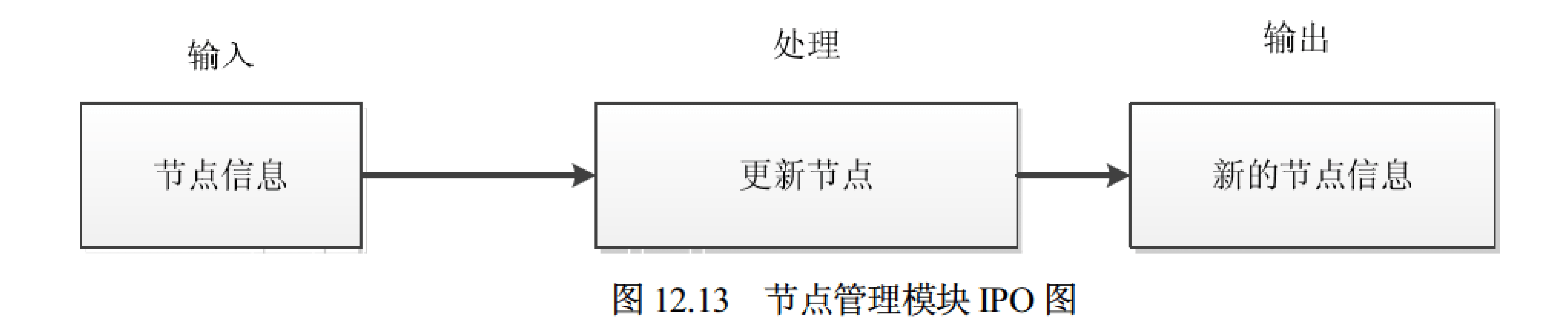


图 12.13　节点管理模块 IPO 图

从图 12.13 可以看出管理员输入管理节点操作信息，节点操作模块根据信息管理节点，完成后返回新的节点信息。

本模块的逻辑流程用时序图表示如图 12.14 所示。

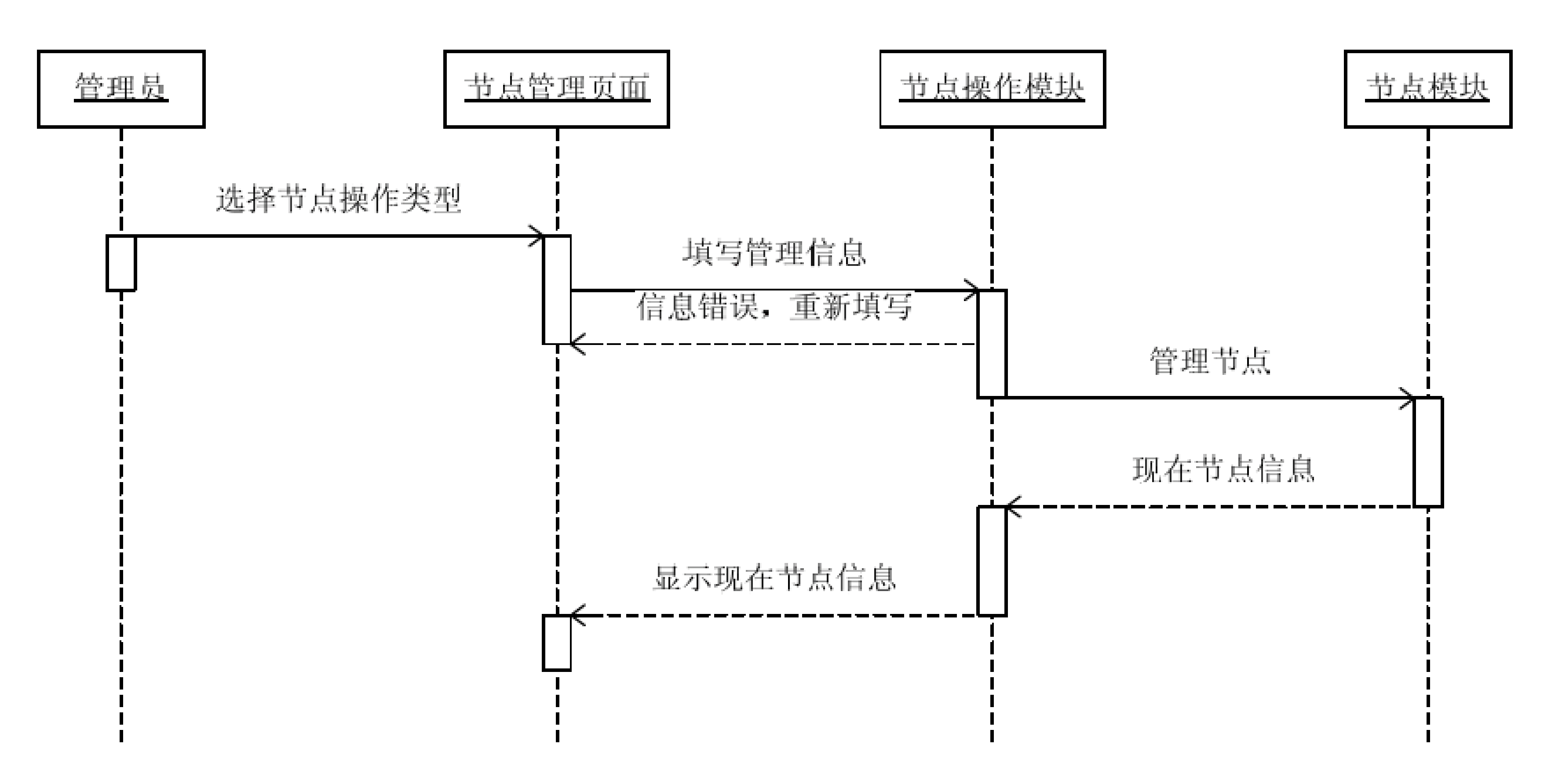


图 12.14　节点管理模块时序图

从图 12.14 可以看出，管理员可以通过选择节点操作的类型、填写新的节点信息，配置系统中的集群，图中给出了明确的操作过程。

（3）添加用户模块设计

系统的用户管理使管理员可以添加系统用户。添加用户模块的 IPO 图表示如图 12.15 所示。



图 12.15　添加用户模块 IPO 图

从图 12.15 可以看出管理员输入用户信息，用户操作模块将用户信息添加到数据库，并返回添加结果。

本模块的逻辑流程用时序图表示如图 12.16 所示。

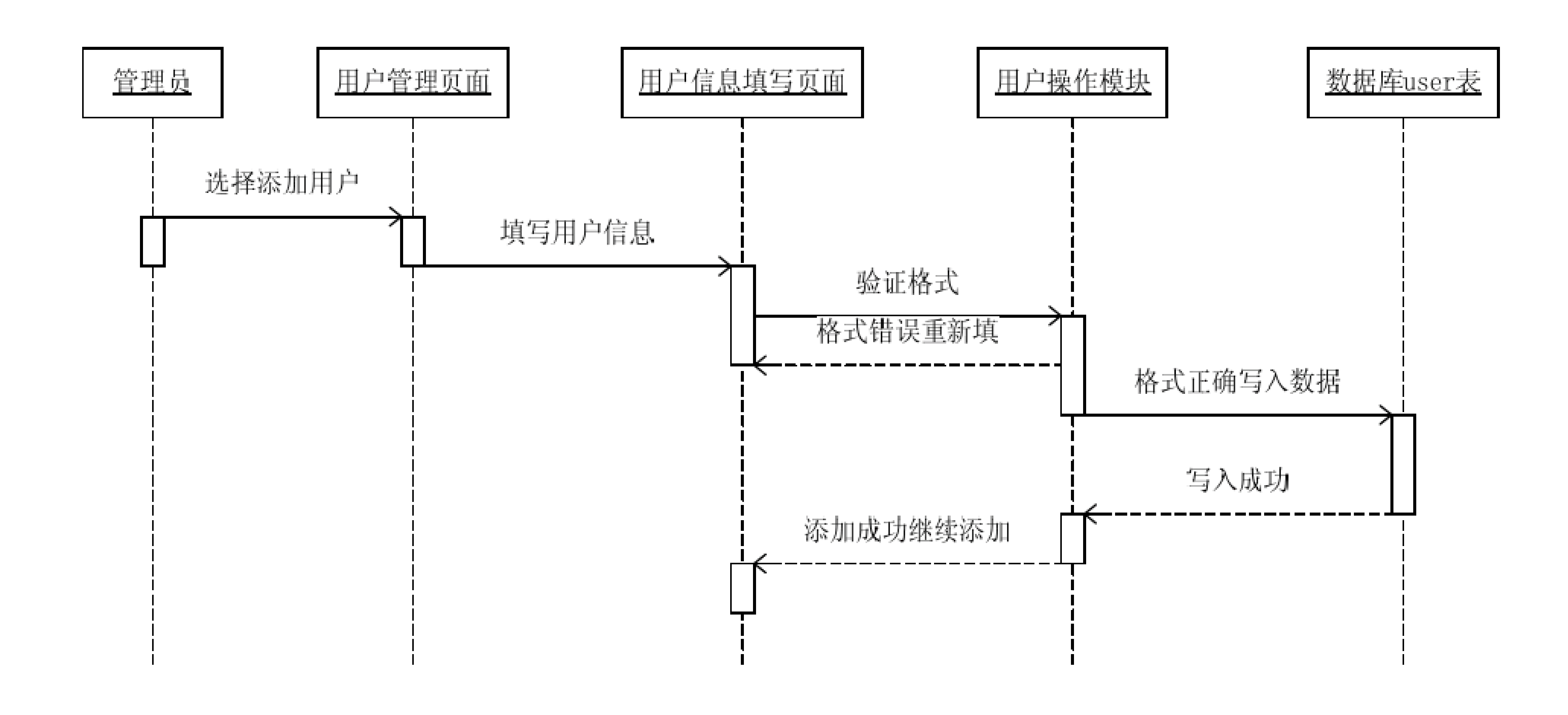


图 12.16　添加用户模块时序图

从图 12.16 可以看出，管理员在用户管理页面选择添加用户然后输入用户信息添加系统用户，图中给出了明确的操作过程。

（4）修改用户信息模块设计

系统的用户管理使管理员可以修改系统的用户信息。修改用户信息模块的 IPO 图表示如图 12.17 所示。

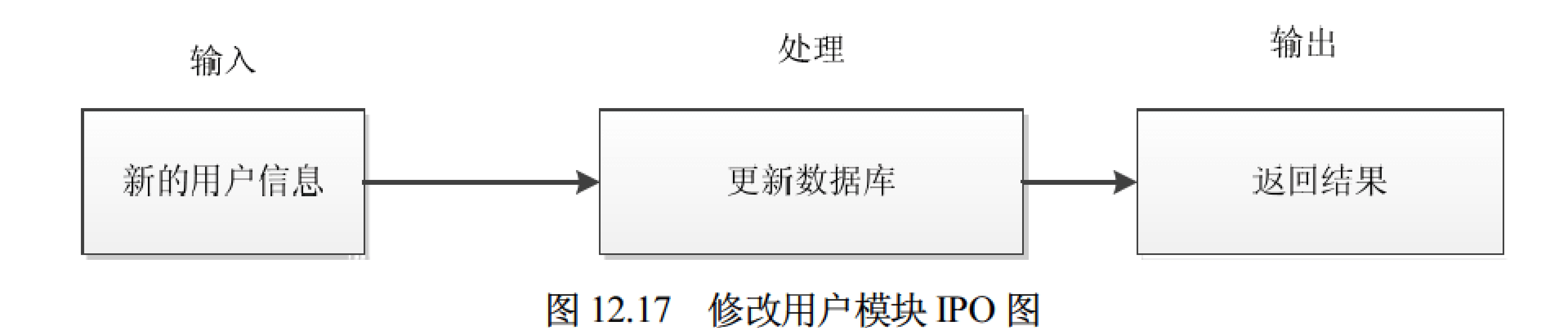


图 12.17　修改用户模块 IPO 图

从图 12.17 可以看出管理员输入新的用户信息，用户操作模块在数据库中更新用户信息，完成后返回更新结果。

本模块的逻辑流程用时序图表示如图 12.18 所示。

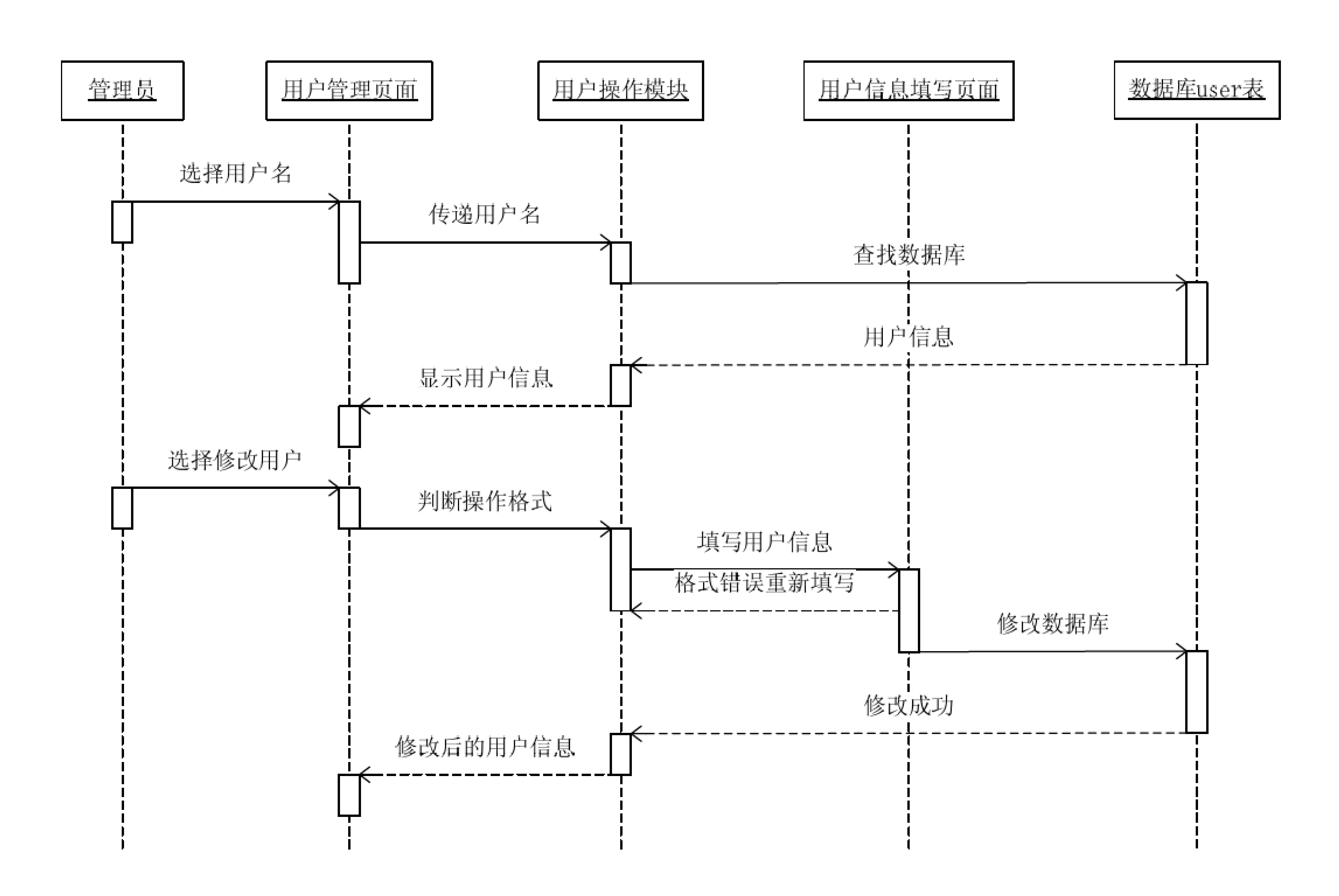


图 12.18　修改用户信息模块时序图

从图 12.18 可以看出，管理员在用户管理页面选择用户然后可以看到用户信息，再选择修改用户信息，进入用户信息修改界面进行修改，图中给出了明确的操作过程。

（5）删除用户模块设计

系统的用户管理使管理员可以删除系统用户。删除用户模块的 IPO 图表示如图 12.19 所示。

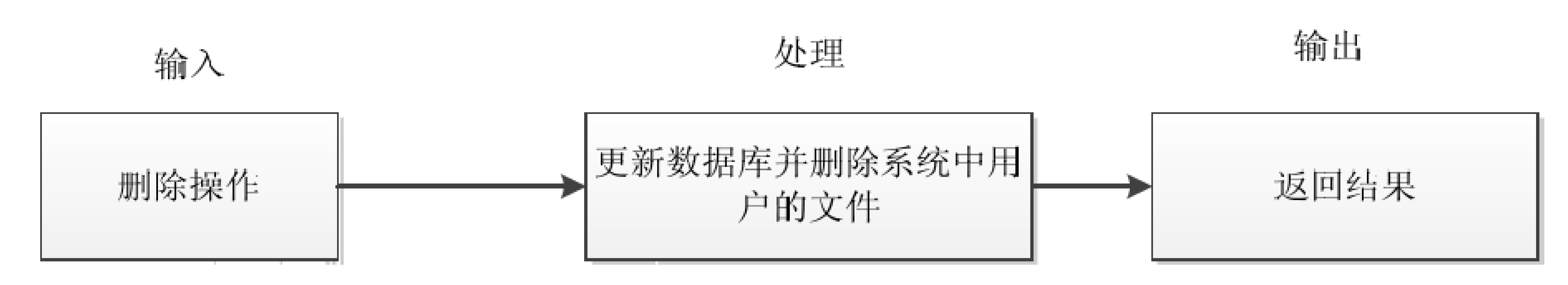


图 12.19　删除用户模块 IPO 图

从图 12.19 可以看出管理员选择删除用户操作，用户操作模块收到信息后删除数据库中用户的信息和该用户的文件，完成后返回操作结果。

本模块的逻辑流程用时序图表示如图 12.20 所示。

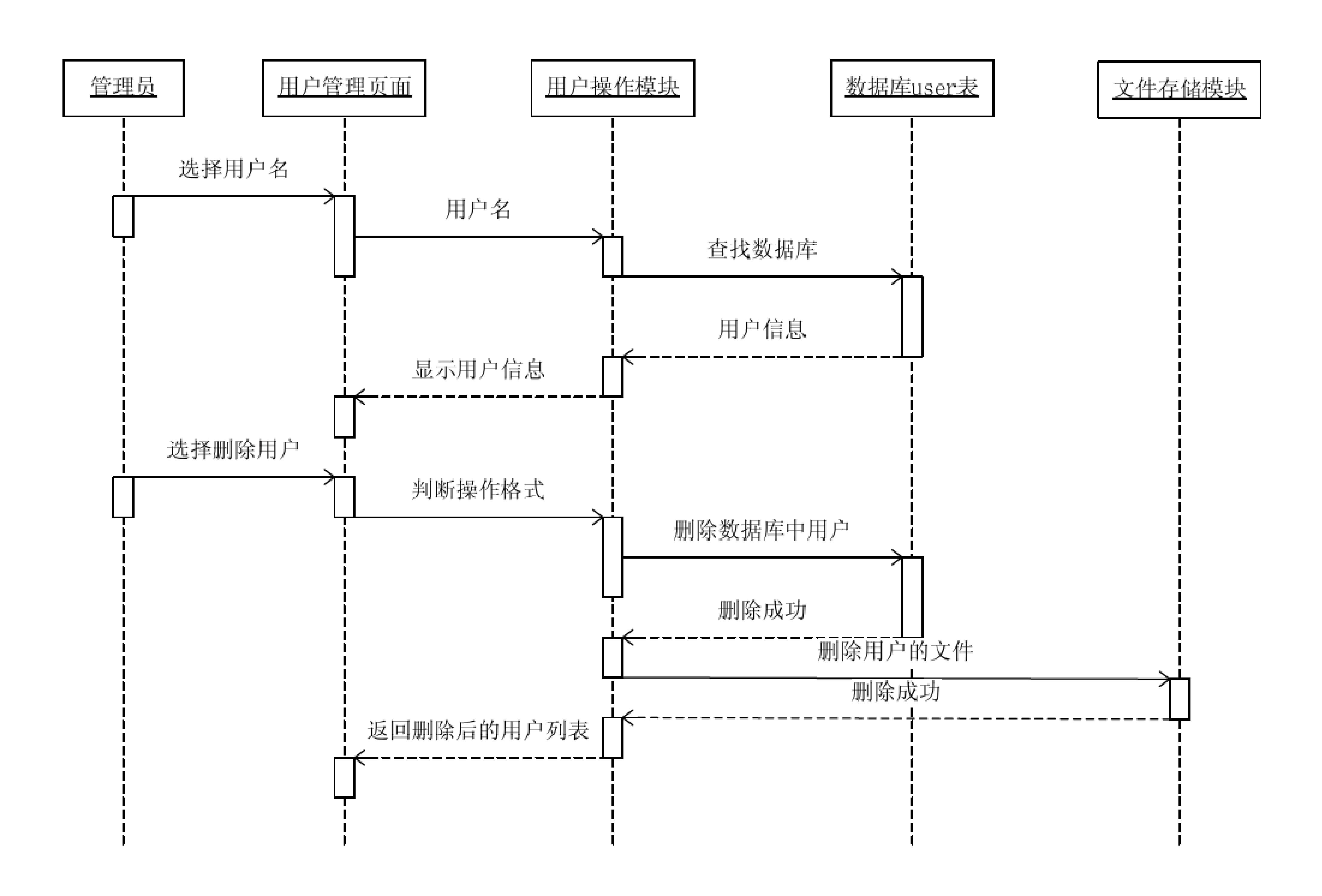


图 12.20　删除用户信息模块时序图

从图 12.20 可以看出，管理员在用户管理页面选择用户然后可以看到用户信息，再选择删除用户，图中给出了明确的操作过程。

3. 用户服务模块

（1）文件上传模块设计

系统的用户可以在用户页面上传自己的文件。文件上传模块的 IPO 图表示如图 12.21 所示。

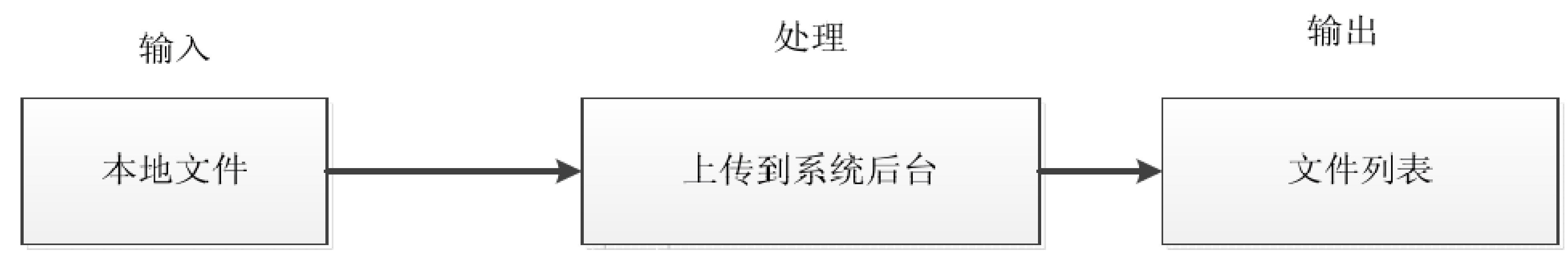


图 12.21　文件上传模块 IPO 图

从图 12.21 可以看出用户选择本地文件，文件操作模块收到信息后将该文件上传到文件系统，完成后返回上传后的文件列表。

本模块的逻辑流程用时序图表示如图 12.22 所示。

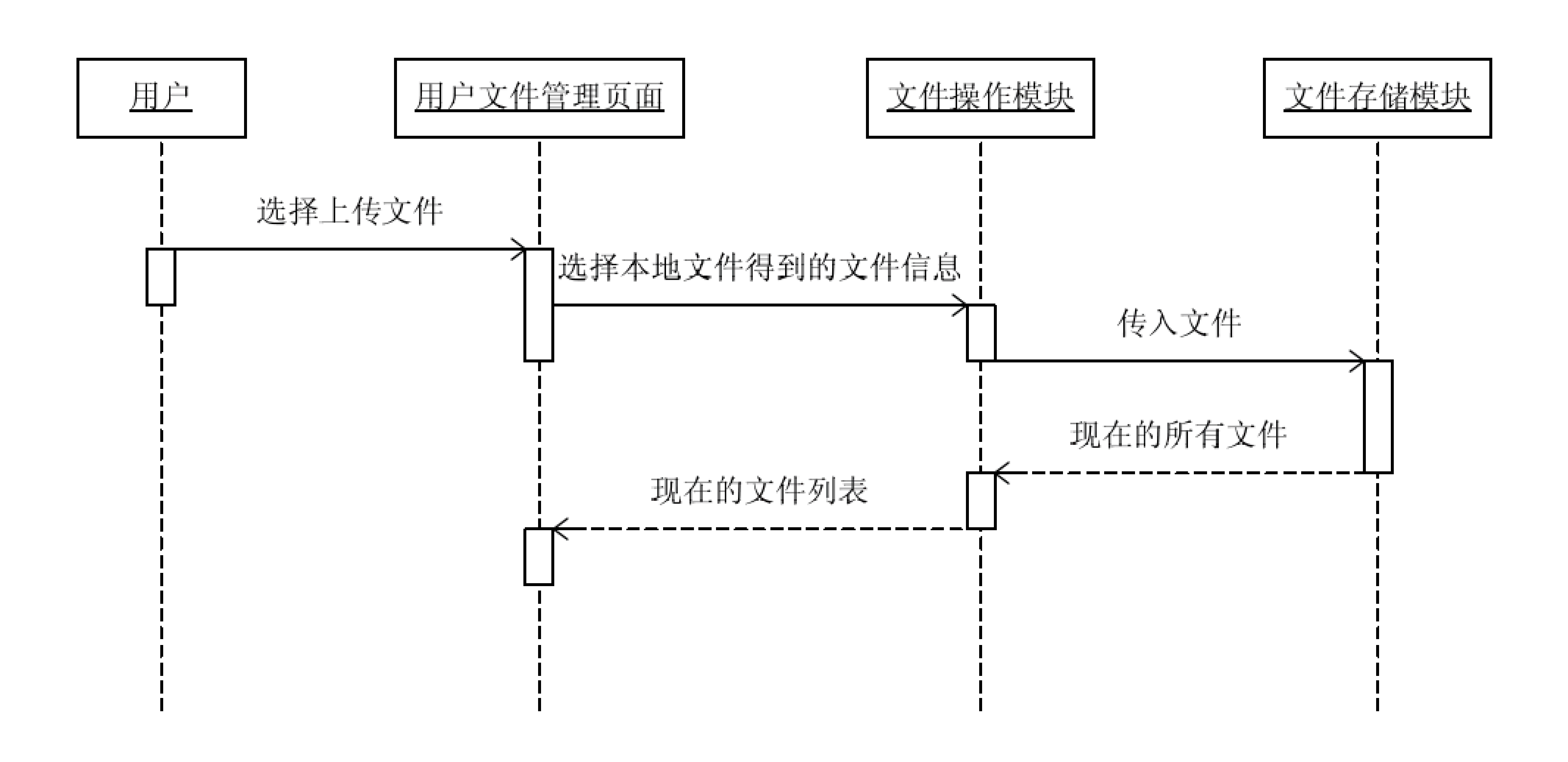


图 12.22　文件上传模块时序图

从图 12.22 可以看出，用户在用户页面选择上传文件进入文件选择框进行文件上传，图中给出了明确的操作过程。

（2）文件下载模块设计

系统的用户可以在用户页面下载自己的文件。文件下载模块的 IPO 图表示如图 12.23 所示。

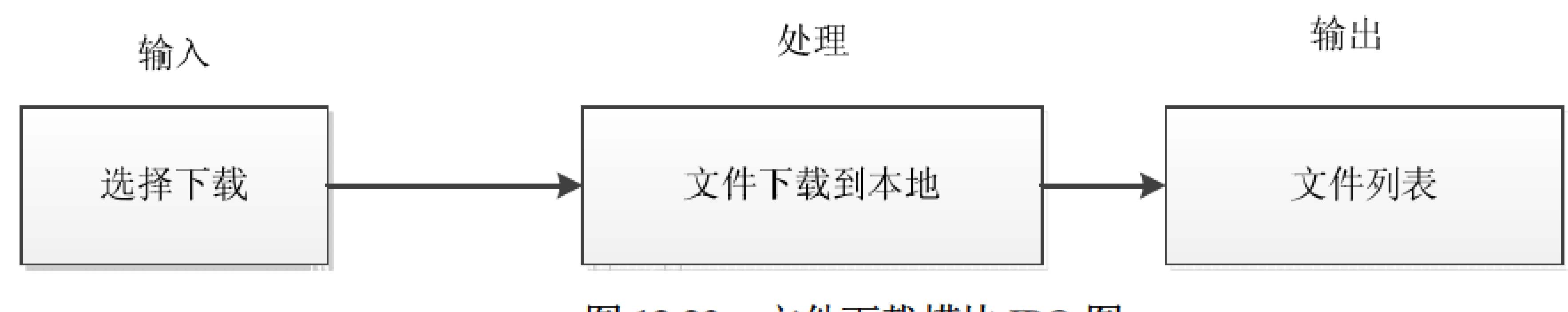


图 12.23　文件下载模块 IPO 图

从图 12.23 可以看出用户选择文件下载，文件操作模块收到下载信息后将该文件下载到用户客户端，完成后返回原文件列表界面，此时用户本地空间的相应位置会有已经下载好的文件。

本模块的逻辑流程用时序图表示如图 12.24 所示。

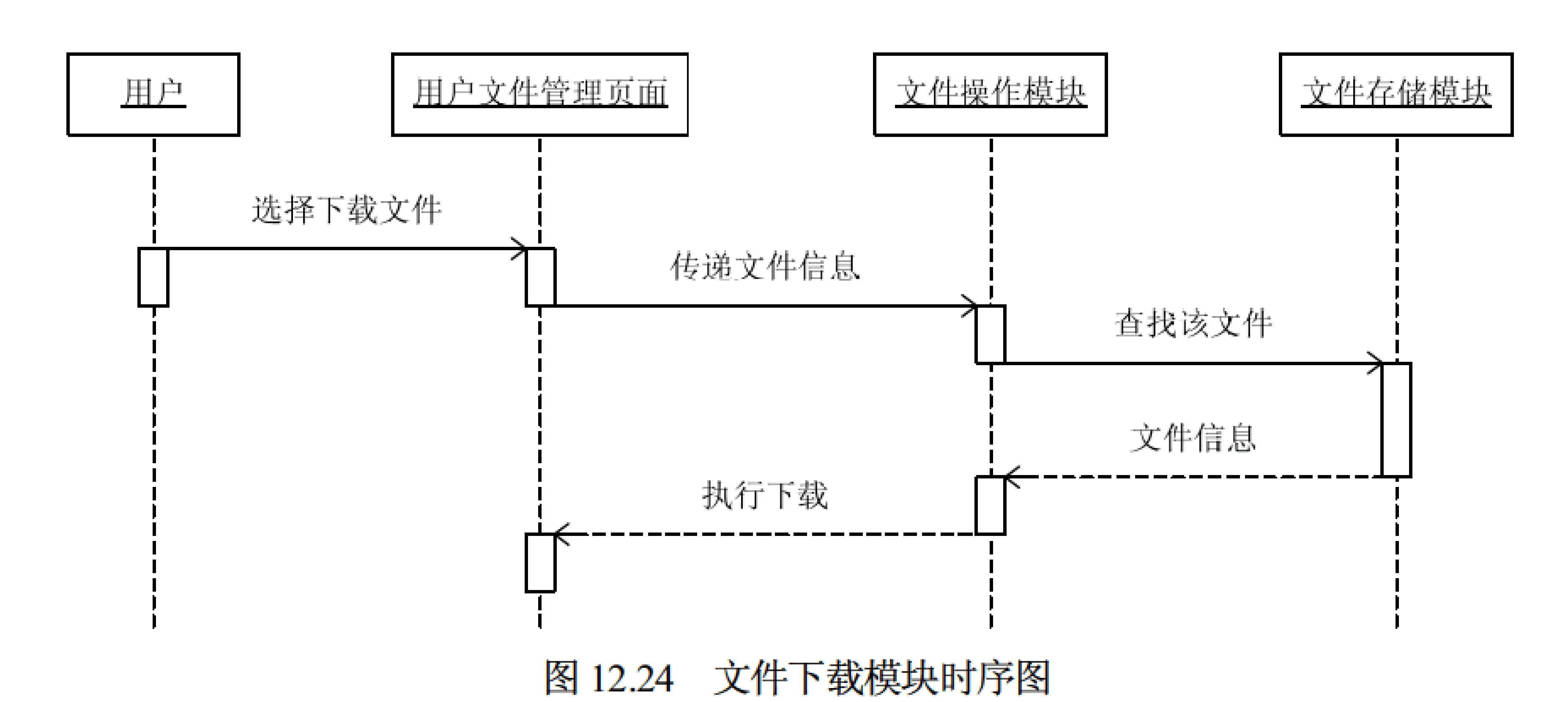


图 12.24　文件下载模块时序图

从图 12.24 可以看出，用户在用户页面选择文件后面的下载文件按钮将调用文件选择存储框选择文件存储的本地位置，选择好后开始下载文件，图中给出了明确的操作过程。

（3）文件重命名模块设计

系统的用户可以在用户页面选择相应文件后面的修改文件名按钮，然后输入新的文件名并提交，进行文件名的修改操作。文件重命名模块的 IPO 图表示如图 12.25 所示。



图 12.25　文件重命名模块 IPO 图

从图 12.25 可以看出用户填写新的文件名，文件操作模块收到信息后将该文件名修改成新的名字，系统验证新文件名的合法性，如果文件名合法则进行修改，完成后返回文件列表，文件名已经被修改；如果文件名不合法系统提示错误信息并返回文件列表，文件名未被修改。

本模块的逻辑流程用时序图表示如图 12.26 所示。

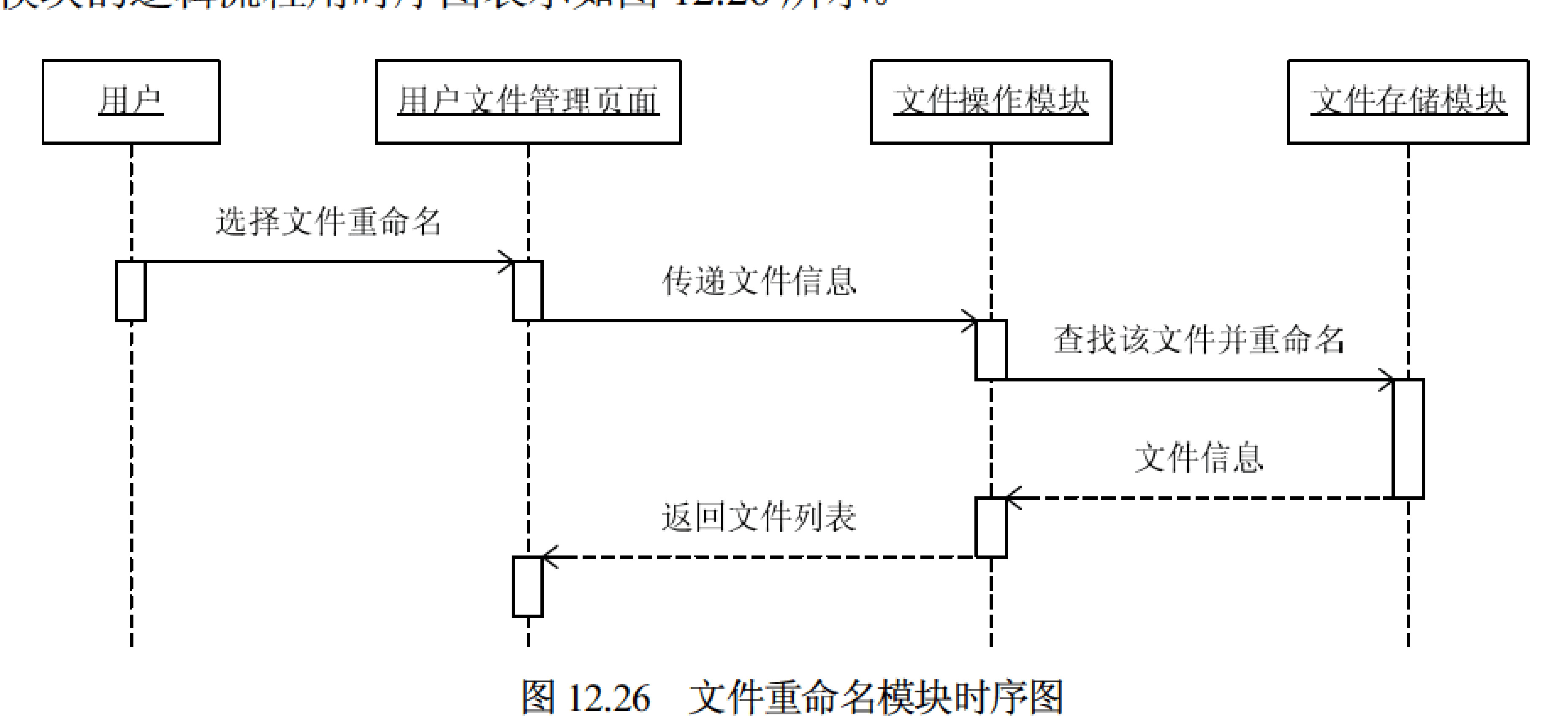


图 12.26　文件重命名模块时序图

从图 12.26 可以看出，用户在用户页面文件后选择重命名修改文件的名称，图中给出了明确的操作过程。

（4）文件搜索模块设计

系统的用户可以在用户页面的搜索框搜索文件或选择左边文件类型搜索文件。文件搜索模块的 IPO 图表示如图 12.27 所示。



图 12.27　文件搜索模块 IPO 图

从图 12.27 可以看出用户输入要查找的文件名，文件操作模块收到信息后对该用户的文件按文件名遍历寻找该文件，完成后返回找到的文件列表。

本模块的逻辑流程用时序图表示如图 12.28 所示。

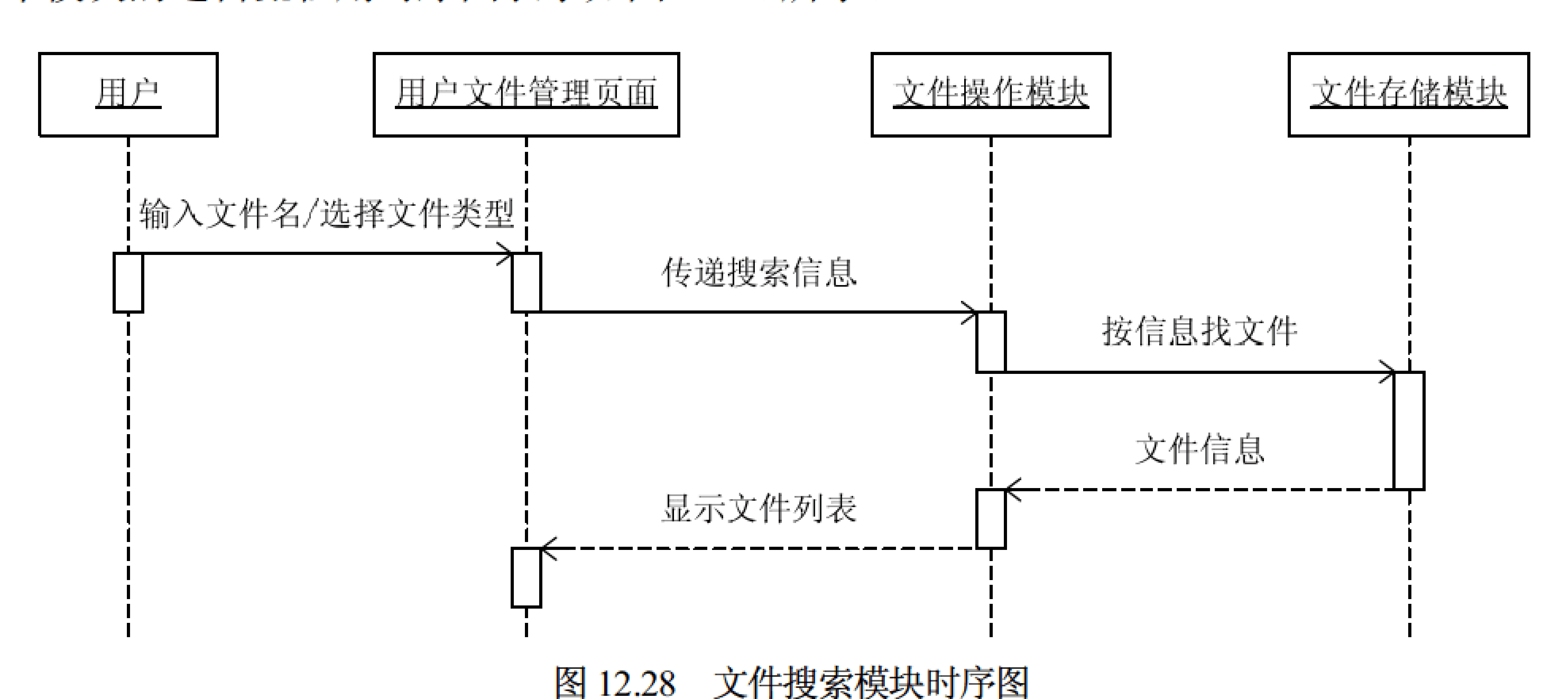


图 12.28　文件搜索模块时序图

从图 12.28 可以看出，用户在搜索框输入关键字，单击搜索进行文件名的模糊搜索；单击页面左边的文件类型使用类型搜索，将按用户所需的文件类型列出所有该类型的文件，图中给出了明确的操作过程。

（5）文件删除模块设计

系统的用户可以在用户页面相应的文件后面单击删除文件按钮将文件删除到回收站。文件删除模块的 IPO 图表示如图 12.29 所示。

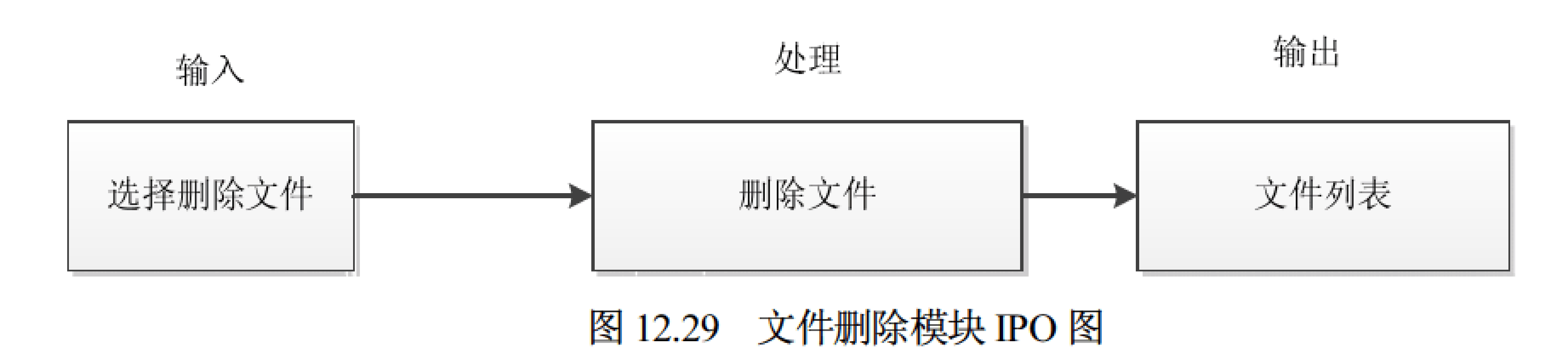


图 12.29　文件删除模块 IPO 图

从图 12.29 可以看出用户选择文件后面的删除按钮，文件操作模块收到信息后对文件执行删除操作，完成后返回文件列表，如果删除成功则返回的文件列表中没有刚删除的文件，否则列表

中该文件还存在。经过测试文件可以成功删除。

本模块的逻辑流程用时序图表示如图 12.30 所示。

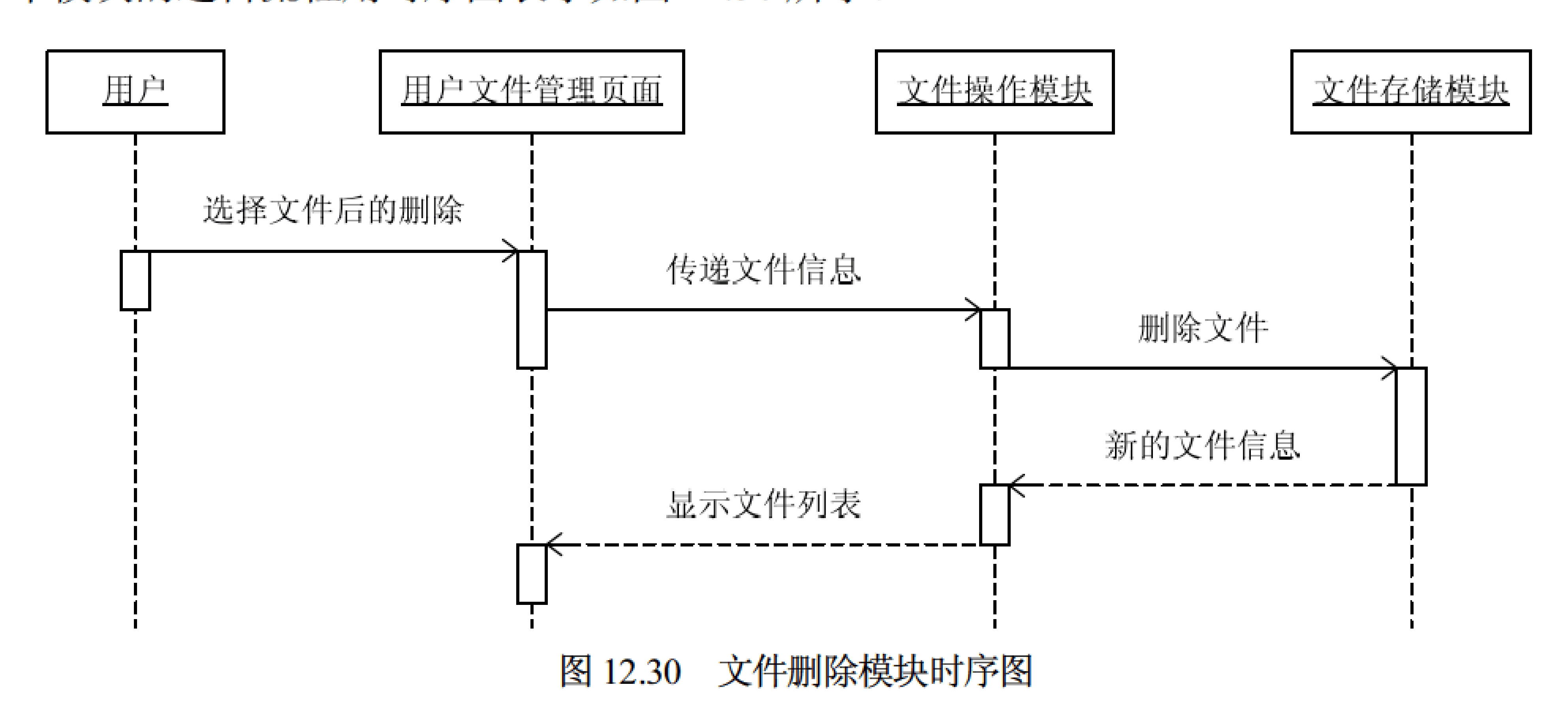


图 12.30　文件删除模块时序图

从图 12.30 可以看出，用户在文件后选择删除文件使用该模块，图中给出了明确的操作过程。

（6）回收站管理模块设计

系统的用户可以在回收站页面管理回收站的文件，可以恢复文件、删除文件、清空回收站。回收站管理模块的 IPO 图表示如图 12.31 所示。

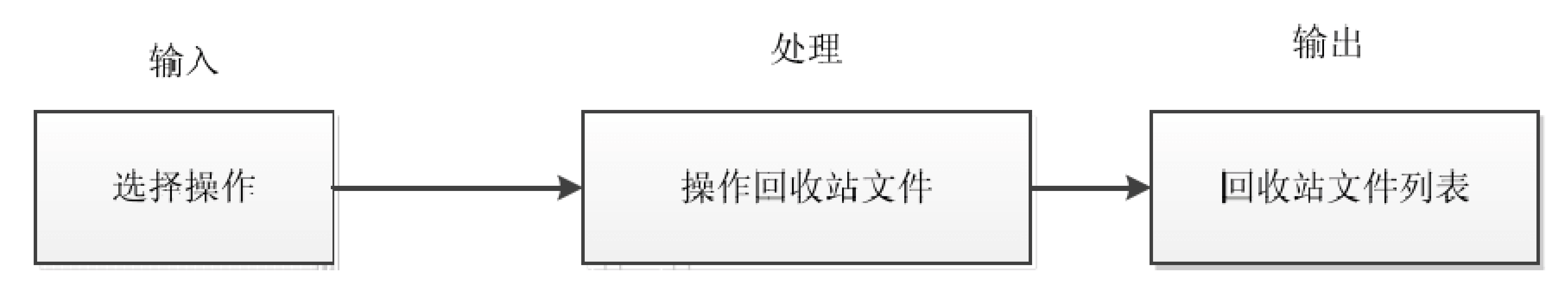


图 12.31　回收站管理模块 IPO 图

从图 12.31 可以看出用户进入回收站后选择文件后面的操作按钮，如果是恢复文件，回收站操作模块对该文件进行恢复操作，如果是删除文件，该模块对文件进行彻底删除，如果用户选择清空回收站则该操作对回收站进行清空操作。对用户的操作执行完成后返回回收站文件列表。

本模块的逻辑流程用时序图表示如图 12.32 所示。

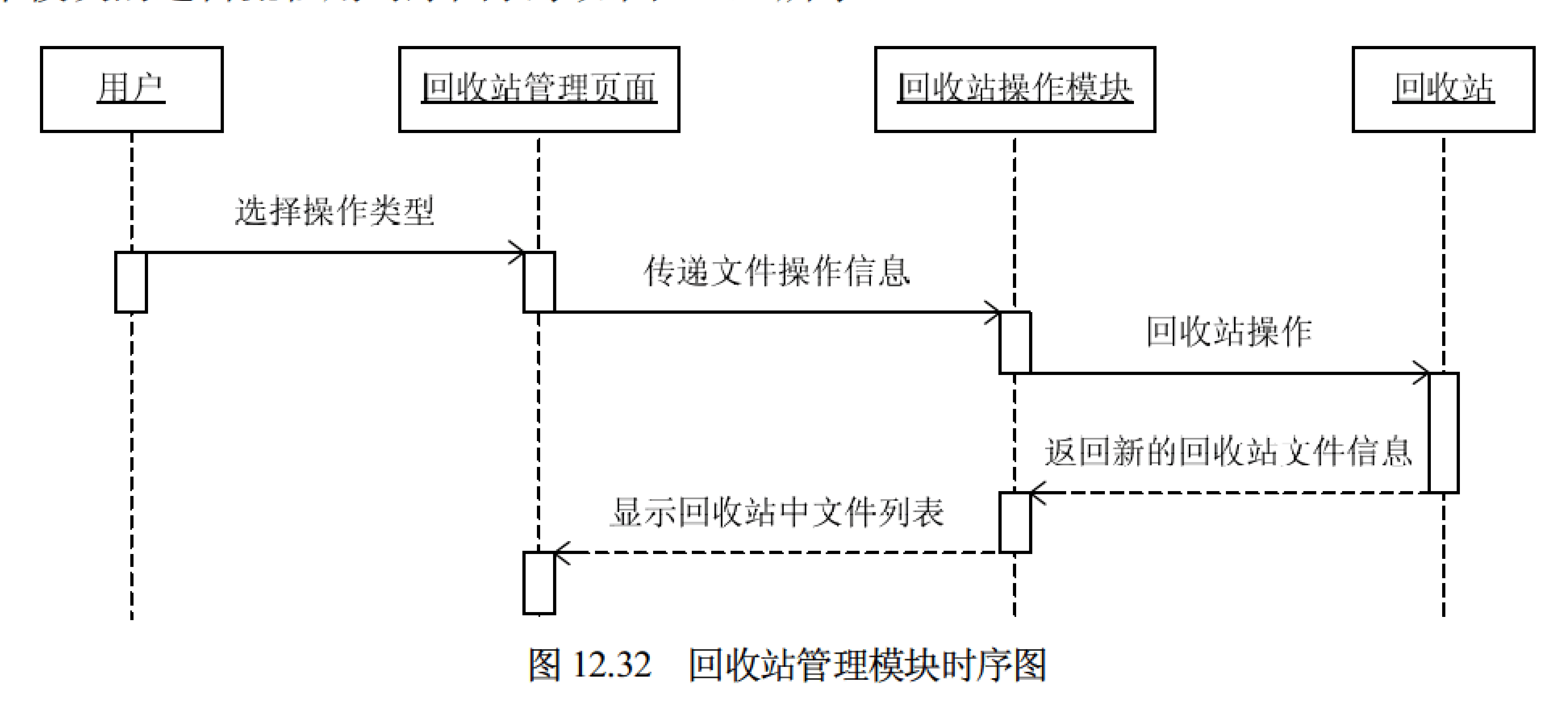


图 12.32　回收站管理模块时序图

从图 12.32 看出，用户在回收站页面选择相应的操作，回收站操作模块收到操作命令后进行回收站管理，图中给出了详细的操作过程。

（7）修改个人信息模块设计

系统的用户可以在用户页面单击修改个人信息按钮进入修改个人信息页面，用户填写合法的新信息提交后，该模块对新的用户信息进行相应的处理。修改个人信息模块的 IPO 图表示如图 12.33 所示。



图 12.33　修改个人信息模块 IPO 图

从图 12.33 可以看出用户在信息修改页面输入新的个人信息提交后，用户操作模块收到信息后核对信息的合法程度，如果信息不合法则在输入框提示错误信息，如果合法则操作模块修改数据库中的用户信息并返回修改成功，完成后返回新的用户信息。

本模块的逻辑流程用时序图表示如图 12.34 所示。

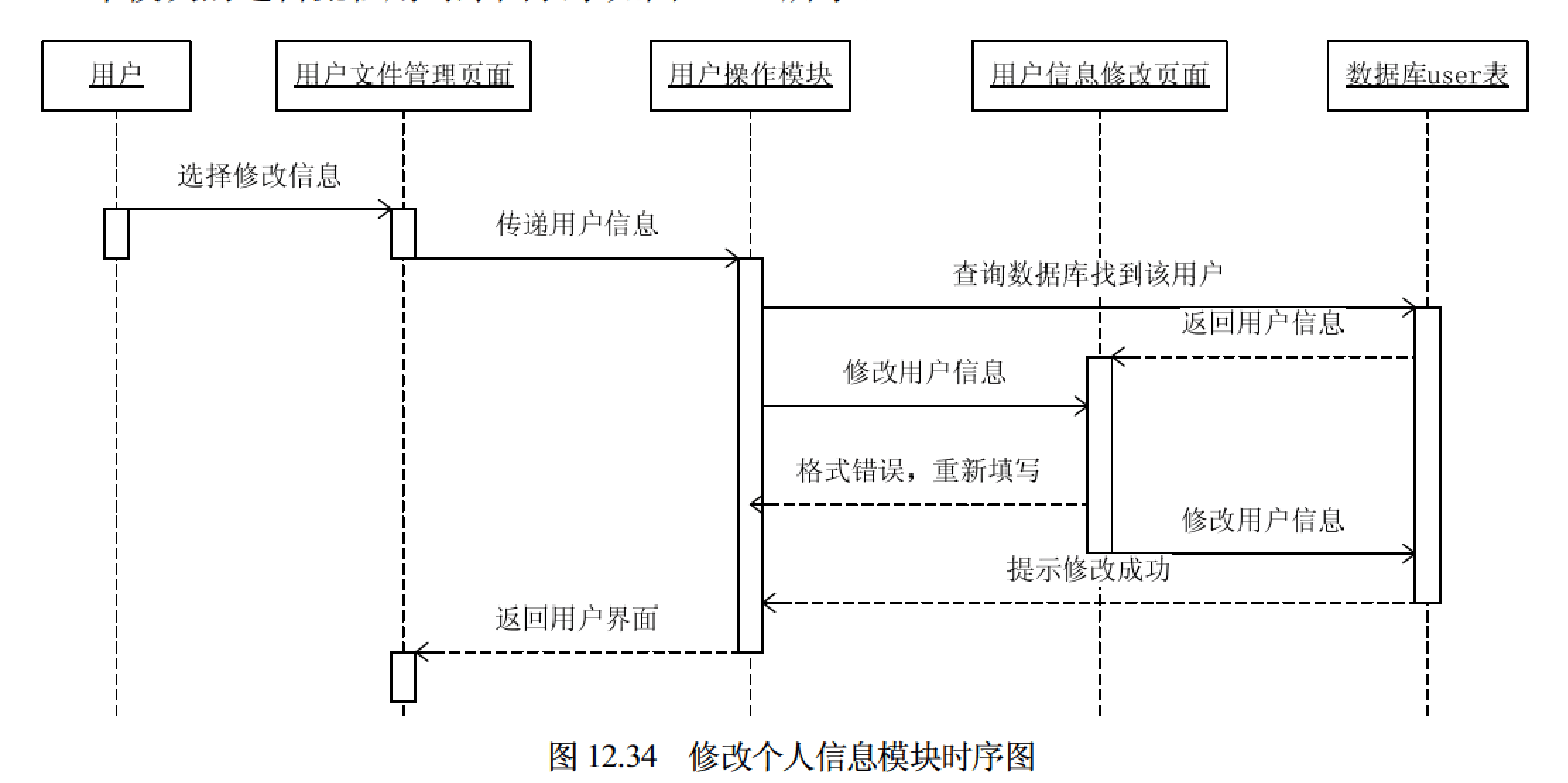


图 12.34　修改个人信息模块时序图

从图 12.34 看出，用户在用户页面选择单击修改个人信息按钮进入个人信息修改模块，图中给出了明确的操作过程。

12.4.3　云文件系统实现

1. 系统存储管理

云文件系统的核心是一个分布式的文件系统。分布式文件系统是指文件系统管理的存储资源

不直接连接在本地结点上，而是通过计算机网络与结点相连。它的设计基于客户机/服务器模式，一个典型的网络可能包括多个供多用户访问的服务器。云文件系统提供了单个访问点和一个逻辑树结构，通过它，用户在访问文件时不需要知道它们的实际位置，分布在多个服务器上的文件在用户面前就如同在网络的同一个位置。另外，可以将同一个网络中的不同计算机上的共享文件夹组织起来，形成一个单独的、逻辑的、层次上的共享的文件系统。云文件系统通过对外的接口让用户不需要了解其内部的复杂机制，而是可以像普通的文件系统一样进行文件的创建、删除、查阅目录信息等类似的操作。下面我们分功能说明该系统的实现。

（1）文件/目录的创建和删除

为了对用户屏蔽实现的复杂性和细节，云文件系统中设计实现了最终面向用户的接口类 FileSystem，它提供了用户能够对文件系统进行各项基本操作的 API。图 12.35 给出了云文件系统中文件/目录的创建过程。

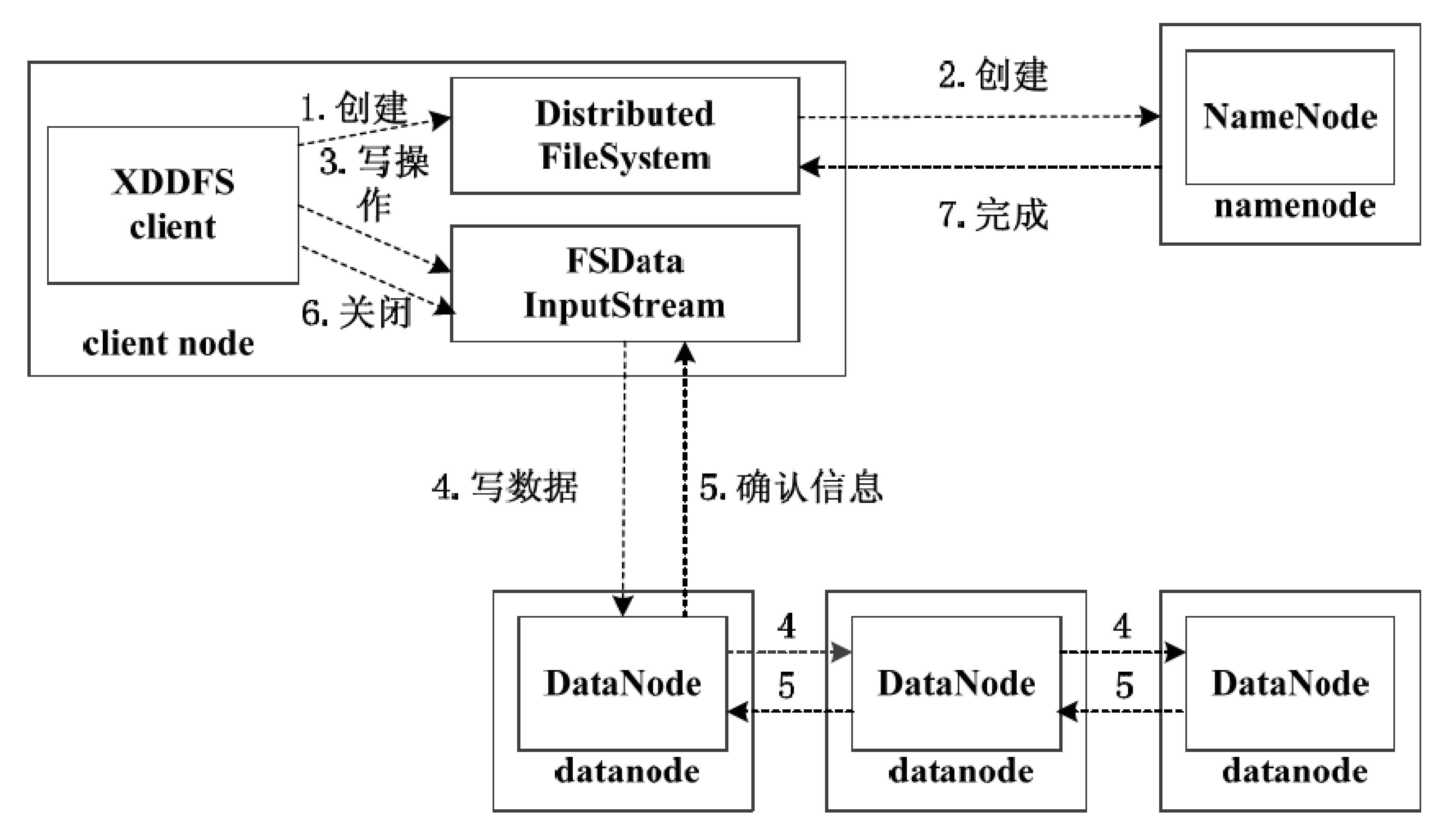


图 12.35　云文件系统创建文件过程图

FileSystem 是一个文件系统的抽象类，针对云文件系统的具体实现类是 DistributedFileSystem。用户想要在云文件系统中创建一个文件或目录时，就不得不提到 XDDFSClient 这个类，它是一个真正实现了客户端功能的类，能够连接到我们的文件系统并执行基本的文件操作。它使用 ClientProtocol 来和 NameNode 通信，并且使用 Socket 直接连接到 DataNode 来完成数据块的读写。FSNamesystem 相当于一本花名册，记录着文件系统的一些元数据，比如从文件名映射到 block 列表。创建文件时要在它这里先“注册”一下。注册的过程有一系列的检查，包括是否处于“安全模式”（系统启动时需要检查数据完整性，不允许写操作）、用户是否有操作权限等。文件的创建主要是通过 FileSystem 的 mkdirs()方法。这个方法在 XDDFSClient 的实例中调用其同名方法 mkdirs()，通过系统的 RPC 机制调用 NameNode 的 mkdir()方法，最终这个调用 Push 到 FSNameSystem 的一个方法，主要是用来检验创建者的访问权限。最后通过 FSDirectory 的方法，构建一个 INodeDirectory 实例添加到文件系统的目录树中。这个时候文件创建操作就算完成了重要的第一步，文件系统中已经有了这个文件的记录。创建文件的序列图如图 12.36 所示。

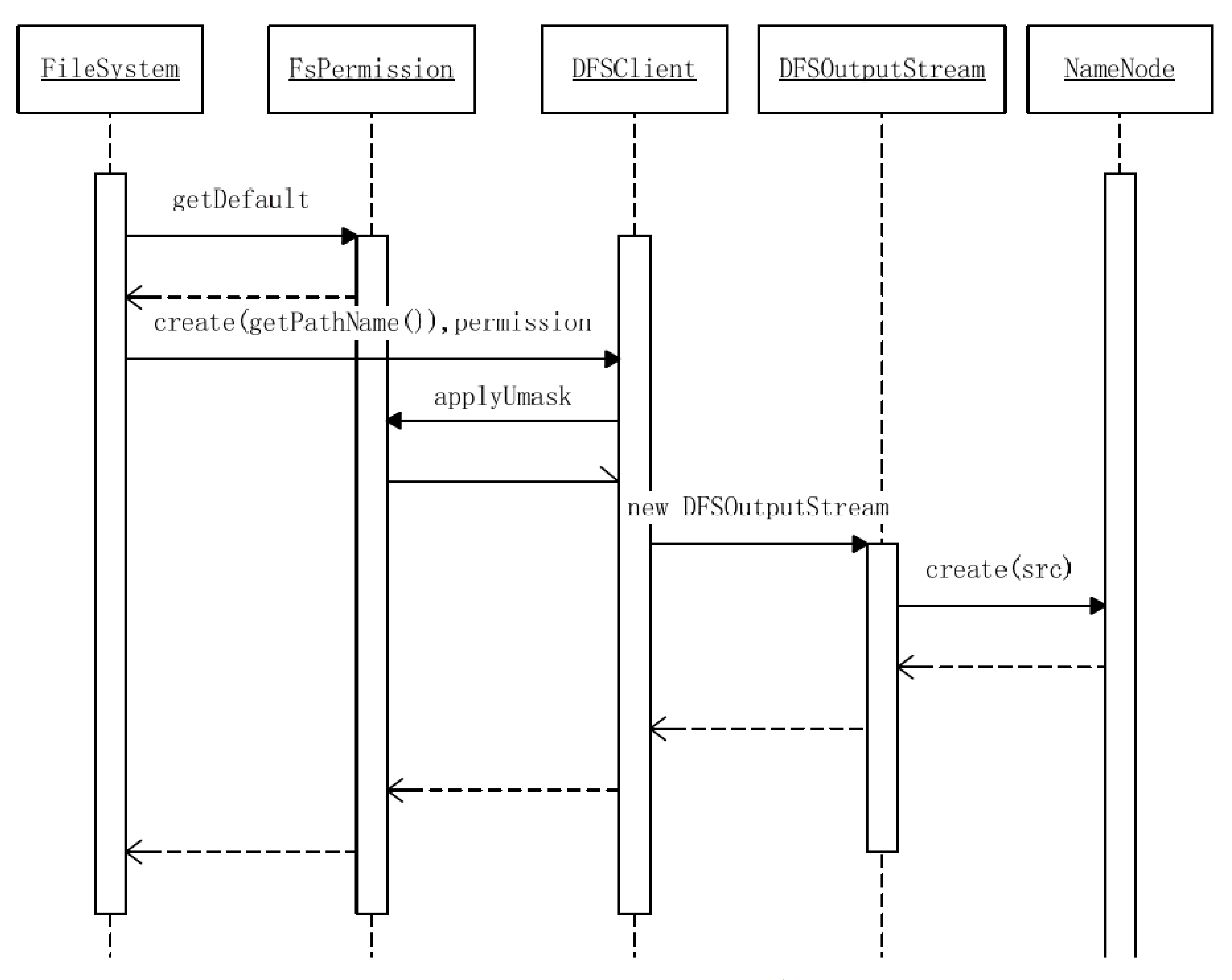


图 12.36　云文件系统文件操作序列图

在删除文件时，有一点需要注意：用户或者应用删除某个文件时，这个文件并没有立刻从 HDFS 中删除。相反，HDFS 将这个文件重命名，并转移到/trash 目录。当文件还在/trash 目录时，该文件可以被迅速地恢复。文件在/trash 中保存的时间是可配置的，当超过这个时间，Namenode 就会将该文件从 namespace 中删除。文件的删除，也将释放关联该文件的数据块。并且，在文件被用户删除和 HDFS 空闲空间的增加之间会有一个等待时间延迟。

下面简单描述这一过程中几个比较重要的类。

- FileSystem

FileSystem 是一个通用文件系统的抽象基类，它可能被实现为分布式文件系统或本地文件系统。在我们的例子中，就有对它的分布式文件系统实现，即 DistributedFileSystem。当一个具体的文件系统被实现时，实现它的具体输入输出流也就不难理解了。在应用中，通过以下方式来获得 FileSystem 的一个实例：

```
FileSystem fs = FileSystem.get(URI.create(uri), conf);
```

- NameNode

NameNode 主要维护文件系统的名字空间和文件的元数据。客户端通过 RPC 机制调用这个类的 create 方法在文件系统中实际创建文件或目录。

ClientProtocol 提供给客户端，用于访问 NameNode。它包含了文件角度上的 HDFS 功能。和

GFS 一样，HDFS 不提供 POSIX 形式的接口，而是使用了一个私有接口。一般来说，程序员通过 org.apache.Hadoop.fs.FileSystem 来和云文件系统打交道，不需要直接使用该接口。

- INode* 类

云文件系统与 Linux 文件系统类似，都使用 inode 来表示一个文件。Namenode 包下有 INode*.Java，类 INode*抽象了文件层次结构。如果我们对文件系统进行面向对象的抽象，可得到如图 12.37 所示的结构图（类 INode*）。

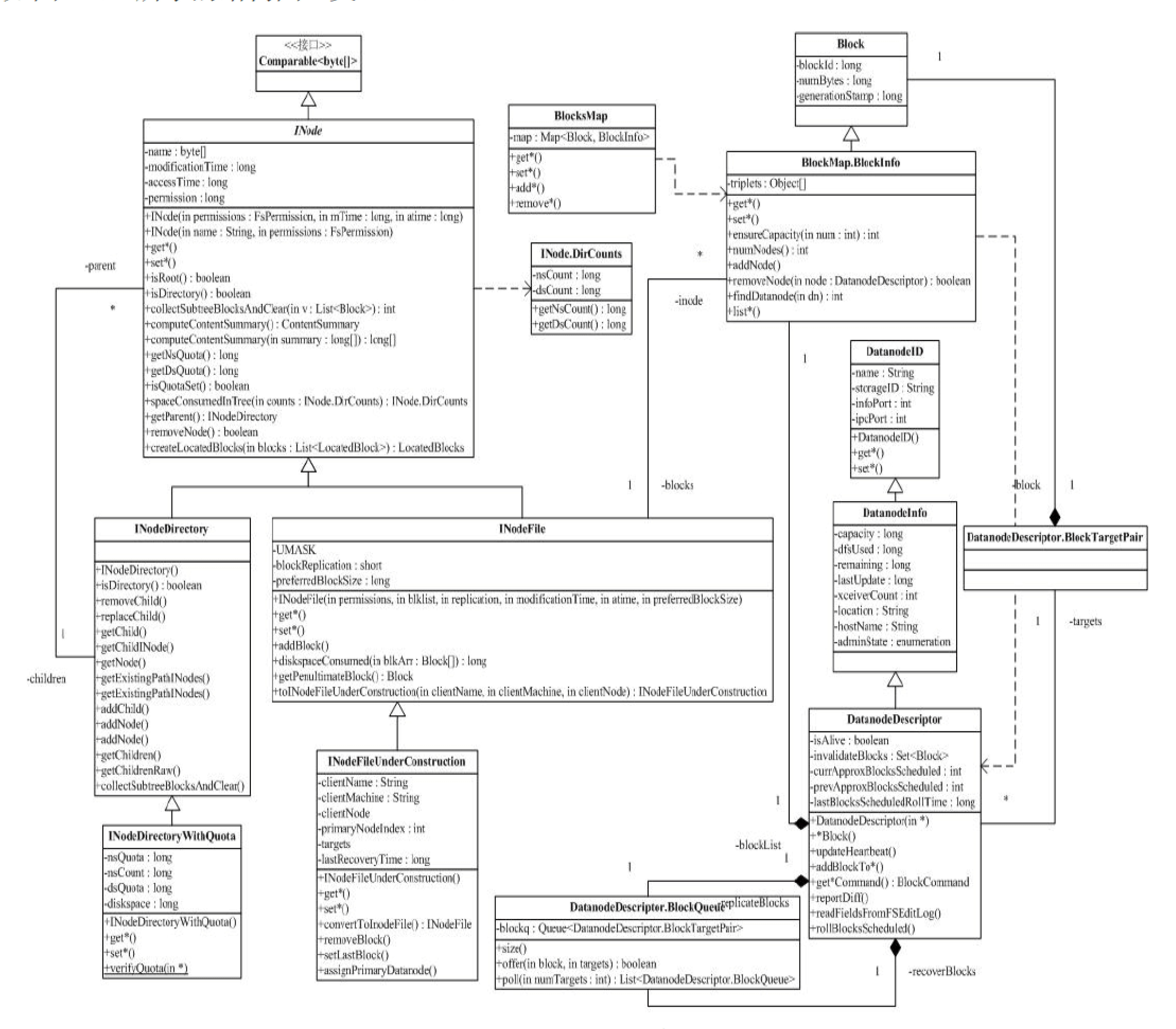


图 12.37　INode*类图

INode 是一个抽象类，它的两个子类，分别对应着目录（INodeDirectory）和文件（INodeFile）。INodeDirectoryWithQuota，如它的名字隐含的，是带容量限制的目录。INodeFileUnderConstruction，抽象了正在构造的文件，当需要在文件系统中创建文件的时候，由于创建过程比较长，目录系统会维护对应的信息。

INode 中的成员变量有：name，目录/文件名；modificationTime 和 accessTime 是最后的修改时间和访问时间； permission 是访问权限。 HDFS 采用了和 UNIX/Linux 类似的访问控制机制。

系统维护了一个类似于 UNIX 系统的组表（group）和用户表（user），并给每一个组和用户一个 ID，permission 在 INode 中是 long 型，它同时包含了组和用户信息。

INode 中存在大量的 get 和 set 方法，当然是对上面提到的属性的操作。导出属性中比较重要的有：collectSubtreeBlocksAndClear，用于收集这个 INode 所有后继中的 Block；computeContentSummary，用于递归计算 INode 包含的一些相关信息，如文件数、目录数、占用磁盘空间。

- XDDFSClient

XDDFSClient 提供了连接到 HDFS 系统并执行文件操作的基本功能。它可以让我们从系统外部了解云文件系统。DFSClient 的成员变量不多，而且大部分是系统的默认配置参数，其中比较重要的是到 NameNode 的 RPC 客户端：

```
public final ClientProtocol namenode;
final private ClientProtocol rpcNamenode;
```

它们的差别是 namenode 在 rpcNamenode 的基础上增加了失败重试功能。方法 ClientDatanodeProtocolProxy()用于生成到 DataNode 的 RPC 客户端。DFSClient 提供了各种 create 方法，它们最后都是构造一个 OutputStream，并将文件名和生成的 OutputStream 加到 leasechecker，完成创建动作。

由以上说明可以知道，用户在客户端通过一个 DFSClient 的实际对象，经过 RPC，实际调用 NameNode 上的服务，完成对云文件系统上文件的创建。创建流程如图 12.38 所示。

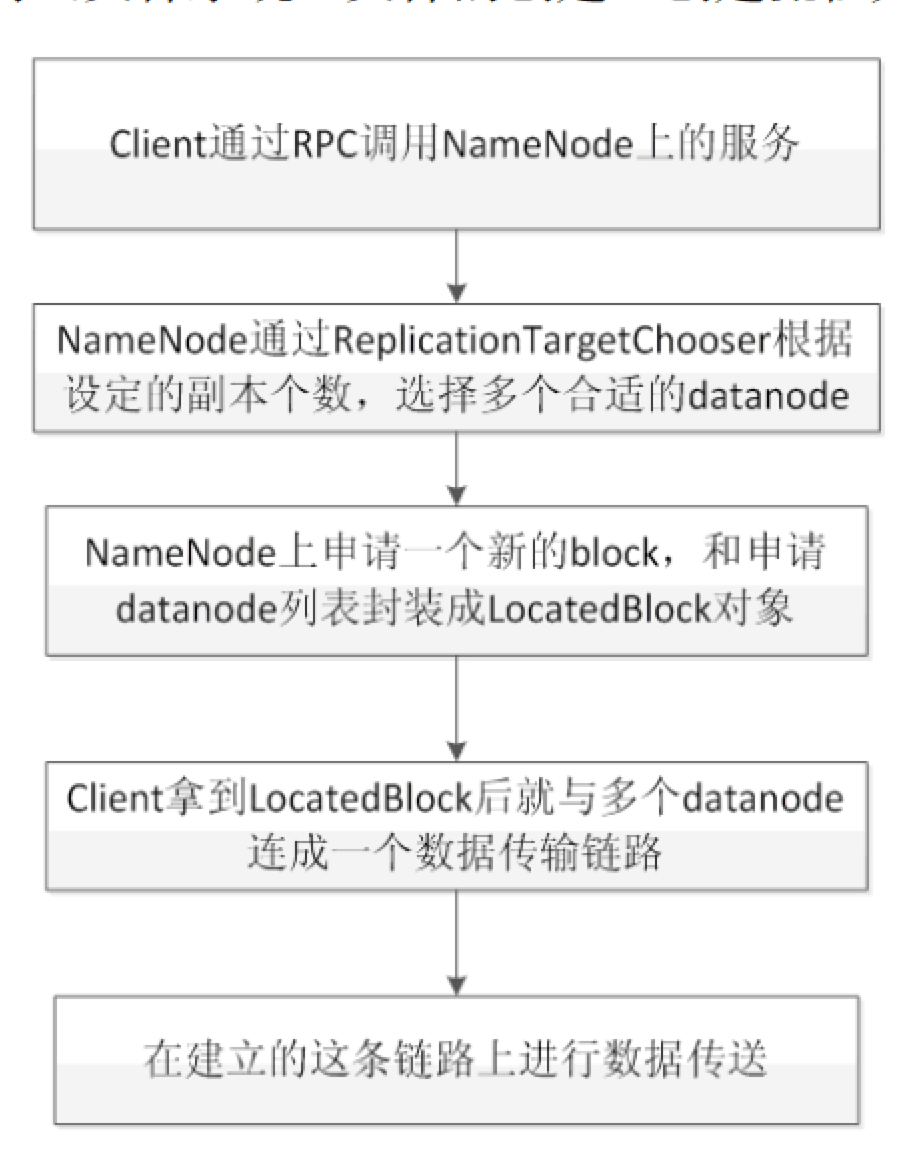


图 12.38　云文件系统创建文件的流程图

（2）文件下载

由于文件存储功能都涉及文件的基本操作，因此涉及的系统中的类和流程也都大同小异，只是具体的功能用到的类的方法不同而已：用户想要对云文件系统有任何操作时，均在本地实例化

出 DFSClient 的一个实际对象，然后该对象 RPC 远程调用 NameNode 的相关方法，对云文件系统进行具体操作即可。因此，下面关于文件系统其他功能的介绍，均仅介绍用到的不同方法，对涉及的类不再赘述。

用户需要将云文件系统中的文件下载到本地时，实例化出云文件系统和本地文件系统的实际对象后，调用 copyToLocalFile()方法即可完成下载的任务。

（3）文件系统在线查看

对于云文件系统来说，如何能够方便地读取系统中的内容是使用者关心的一个问题。云文件系统提供了一个方便的文件访问接口以进行文件系统内容的读取。这里主要通过 HDFS 提供的 API 进行文件读取。

HDFS 主要通过 FileSystem 类来完成对文件的打开操作。和 Java 使用 Java.io.File 来表示文件很不相同，HDFS 中的文件是通过 Path 类来表示的。FileSystem 通过静态方法 get(Configuration conf)来获得 FileSystem 的实例。通过该实例，可以使用 FileSystem 的 open、seek 等方法来实现对文件系统的访问。

通过 FileSystem 的源代码可以看到，最终 open 方法落到一个抽象方法来实现文件的打开，具体的实现方式由继承自 FileSystem 的具体文件系统的实现来决定。

2. 系统监控功能

（1）系统状态展示

先看一下图 12.39 的 NameNode 的类属关系图。

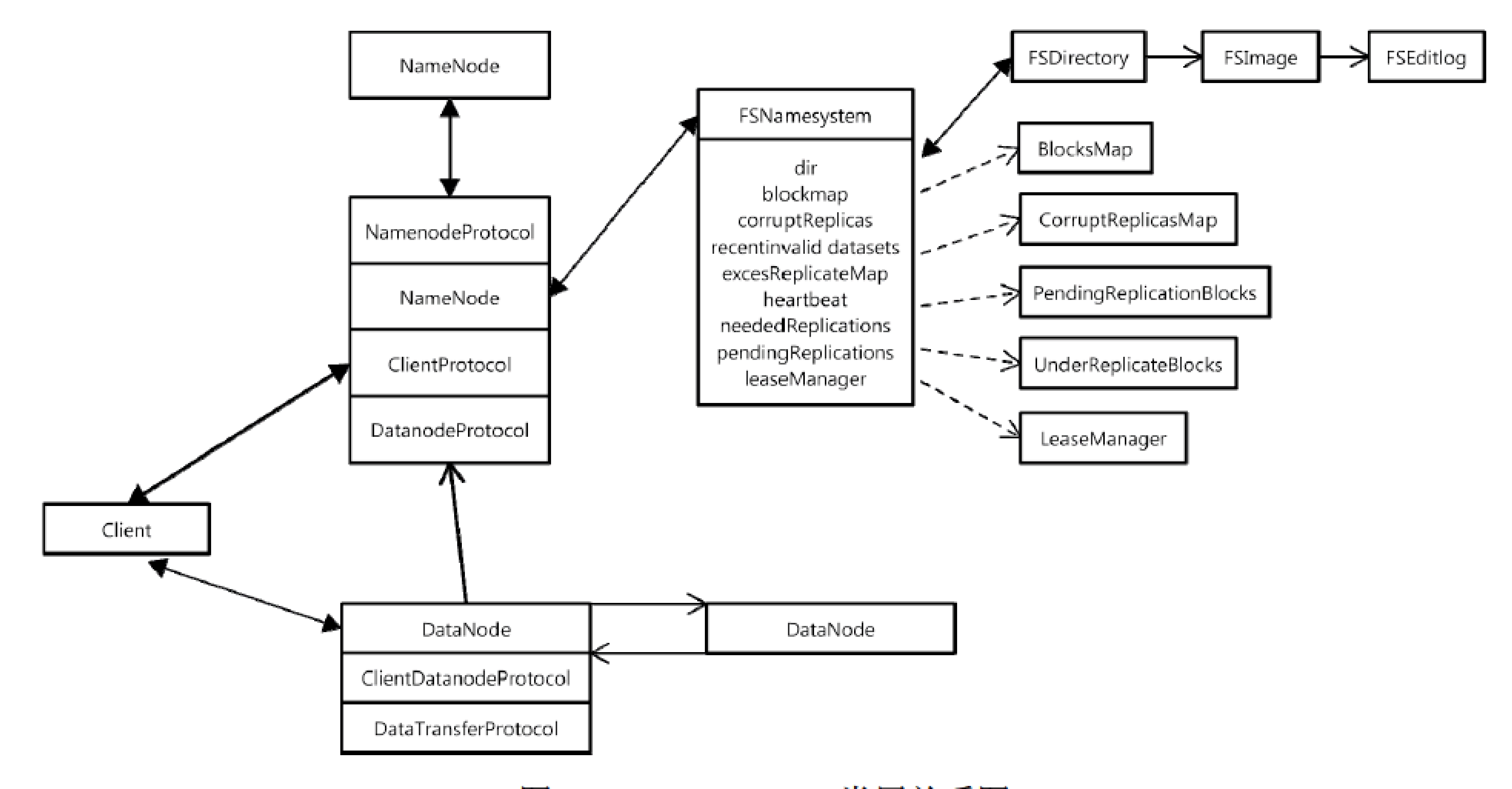


图 12.39　NameNode 类属关系图

NameNode 对象中保存了 HDFS 运行状态信息，在 NameNode 启动时，信息进行更新和封装，同时 NameNode 对象在服务器的 Application 中持久存在，只要通过自身对象的 get 方法就可以得到当前状态，如从 nameNodeAddress 中得到当前主机地址（hostName）和端口号（port）等。VersionInfo 保存了版本信息和外部环境，可用于查询运行时间、当前用户、版本号等内容。

旧版本中只存在两种类型的 NameNode，即主从 NameNode，其中从 NameNode 用来周期性地创建检查点。但在新版本中有 4 种 NameNode 类型，分别为：

- active node，即通常所说的主 NameNode。
- backup node，完成备份功能。
- checkpoint node，周期性地创建检查点。
- standby node，独立的 NameNode。

（2）查看 NameNode 日志

FSEditLog.Java 类提供了 NameNode 操作日志和日志文件的相关方法，相关类图如图 12.40 所示。

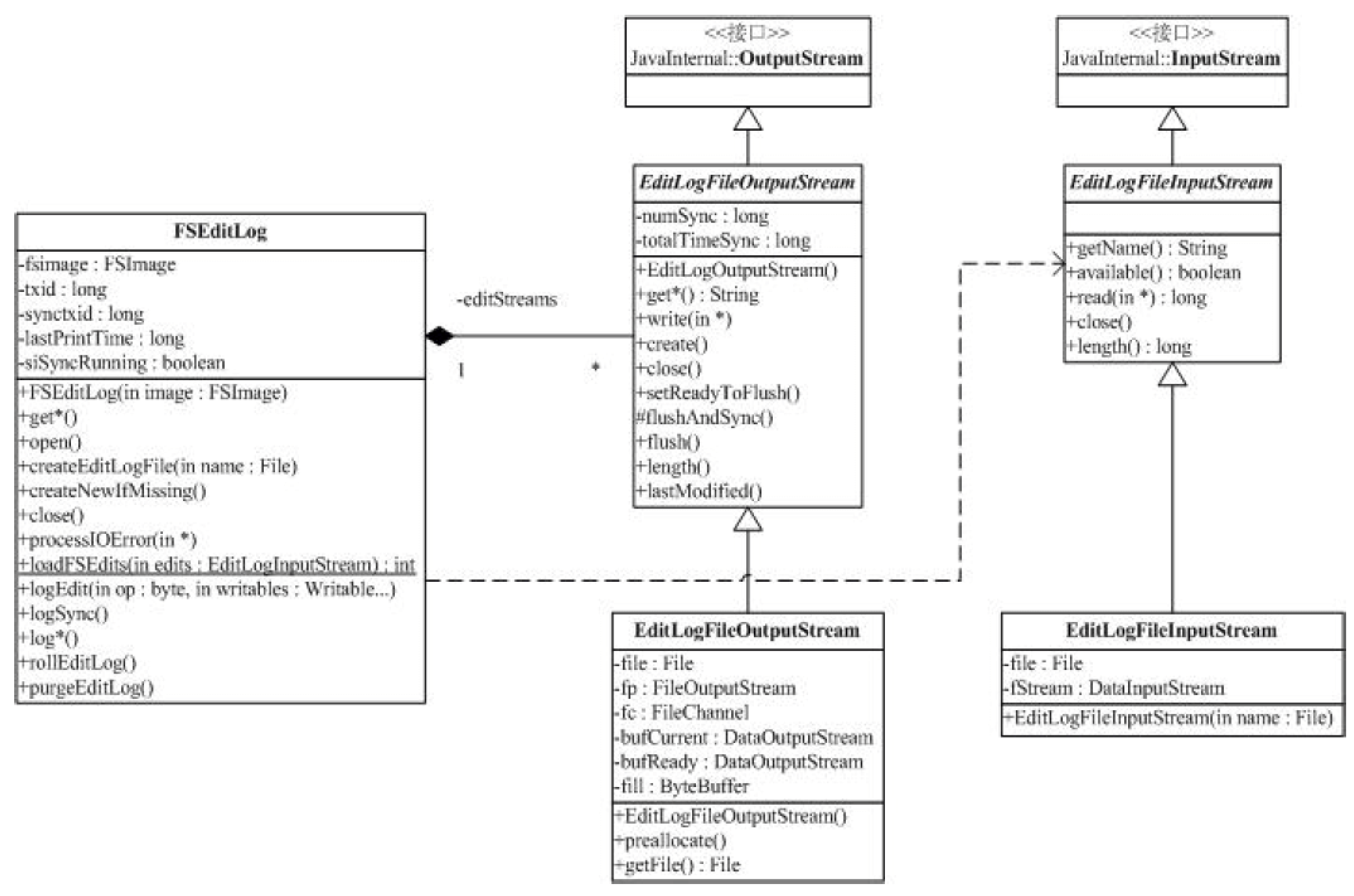


图 12.40　云文件系统操作日志文件相关类图

首先是 FSEditLog 依赖的输入/输出流。输入流基本上没有新添加功能；输出流在打开的时候，会写入日志的版本号（最前面的 4 字节），同时，每次将内存刷到硬盘时，会为日志尾部写入一个特殊的标识（OP_INVALID）。

FSEditLog 有打开/关闭的方法，它们都是很简单的方法，就是关闭的时候，要等待所有正在写日志的操作都完成以后，才能关闭。processIOError 用于处理 IO 出错，一般这会导致 Storage 的日志文件被关闭，如果系统再也找不到可用的日志文件，NameNode 将会退出。

loadFSEdit 负责读取日志文件，并把日志应用到内存中的目录结构中。它又调用了 loadEditRecords，在该方法中完成实际的读取。每一个操作用一个字节常量来表示，比如 OP_ADD、OP_MKDIR 和 OP_CLOSE 等操作。每读取一个操作标识 OP_*，接着读取其后的参数，再调用

FSDirectory 中对应的 unprotected*方法。

logEdit 的作用和 loadFSEdits 相反，它向日志文件中写入日志记录。对于那些需要写日志的操作，这里暂举几例，下面的是它们的参数。

- logOpenFile（OP_ADD）：申请 lease

path（路径）/replication（副本数，文本形式） /modificationTime（修改时间，文本形式）/accessTime（访问时间，文本形式）/preferredBlockSize（块大小，文本形式）/BlockInfo[]（增强的数据块信息，数组）/permissionStatus（访问控制信息）/clientName（客户名）/clientMachine（客户机器名）

- logCloseFile（OP_CLOSE）：归还 lease

path/replication/modificationTime/accessTime/preferredBlockSize/BlockInfo[]/permissionStatus

- logMkDir（OP_MKDIR）：创建目录

path/modificationTime/accessTime/permissionStatus

- logRename（OP_RENAME）：改文件名

src（原文件名）/dst（新文件名）/timestamp（时间戳）

- logSetReplication（OP_SET_REPLICATION）：更改副本数

src/replication

- logSetQuota（OP_SET_QUOTA）：设置空间额度

path/nsQuota（文件空间额度）/dsQuota（磁盘空间额度）

- logSetPermissions（OP_SET_PERMISSIONS）：设置文件权限位

src/permissionStatus

- logSetOwner（OP_SET_OWNER）：设置文件组和主

src/username（所有者）/groupname（所在组）

- logDelete（OP_DELETE）：删除文件

src/timestamp

- logGenerationStamp（OP_SET_GENSTAMP）：文件版本序列号

genstamp（序列号）

- logTimes（OP_TIMES）：更改文件更新/访问时间

src/modificationTime/accessTime

这些方法调用了 logEdit 方法。通过上面的分析，清楚地展示了日志文件里记录了哪些信息。

rollEditLog()用于关闭 edits，打开日志到 edits.new。purgeEditLog()的作用正好相反，它删除老的 edits 文件，然后把 edits.new 改名为 edits。这也是 Hadoop 在做更新修改时经常采用的策略。

（3）文件系统目录结构展示

通过首页可以连接到云文件系统，查看云文件系统结构和框架，浏览每层文件的区块信息、权限，创建用户和集群等。要获得这些信息，需要得到请求的 DFSClient。DFSClient 是一个真正实现了客户端功能的类，它能够连接到一个 Hadoop 文件系统并执行基本的文件任务。它使用 ClientProtocol 来和 NameNode 通信，并且使用 Socket 直接连接到 DataNode 来完成块数据的读/写。Hadoop 用户应该得到一个 DistributedFileSystem 实例，该实现使用 DFSClient 来处理文件系统任务，而不是直接使用 DFSClient。DFSClient 类图如图 12.41 所示。

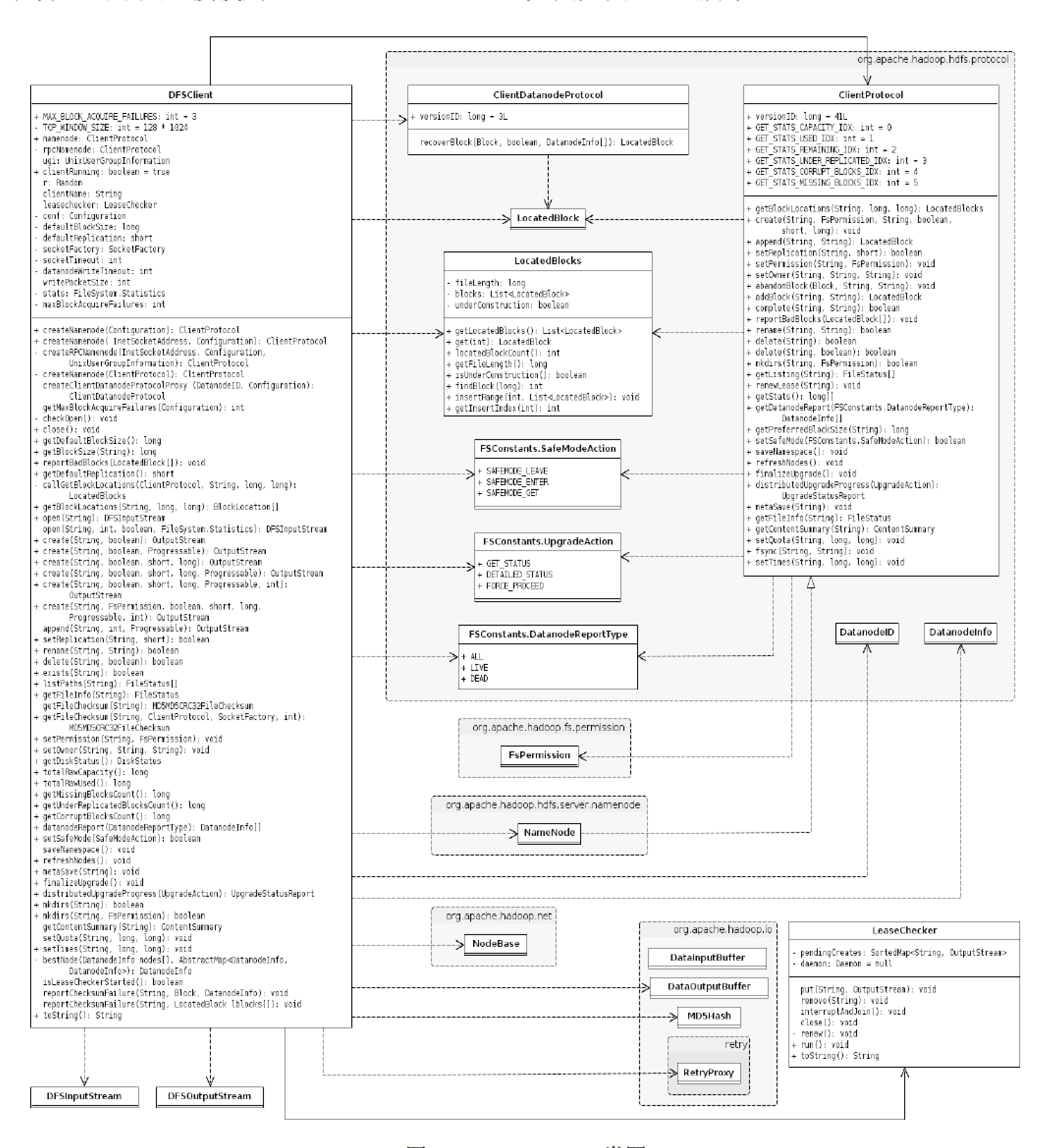


图 12.41　DFSClient 类图

DFSClient 一些重要的属性如下。

- MAX_BLOCK_ACQUIRE_FAILURES：块最大请求失败次数，值为 3。
- TCP_WINDOW_SIZE：TCP 窗口的大小，值为 128KB，在 seek 操作中会用到，假如目标位置在当前块内及在当前位置之后，并且与当前位置的距离不超过 TCP_WINDOW_SIZE，那么这些数据很可能在 TCP 缓冲区中，只需要通过读取操作来跳过这些数据。
- rpcNamenode：通过建立一个 RPC 代理来和 namenode 通信。
- namenode：在 rcpNamenode 基础上封装了一个 Retry 代理，添加了一些 RetryPolicy。
- leasechecker：租约管理，用于管理正被写入的文件输出流。
- defaultBlockSize：块大小，默认是 64MB。
- defaultReplication：副本数，默认是 3。
- socketTimeout：socket 超时时间，默认是 60s。
- datanodeWriteTimeout：datanode 写超时时间，默认是 480s。
- writePacketSize：写数据时，一个 packet 的大小，默认是 64KB。
- maxBlockAcquireFailures：块最大请求失败次数，默认是 3，主要用于向 datanode 请求块时，失败了可以重试。

DFSClient 的成员变量不多，而且大部分是系统的默认配置参数，其中比较重要的是到 NameNode 的 RPC 客户端：

```
public final ClientProtocol namenode;
final private ClientProtocol rpcNamenode;
```

它们的差别是 namenode 在 rpcNamenode 的基础上，增加了失败重试功能。DFSClient 中提供各种构造它们的 static 函数，createClientDatanodeProtocolProxy 用于生成到 DataNode 的 RPC 客户端。DFSClient 的构造函数也比较简单，就是初始化成员变量，close 用于关闭 DFSClient。DFSClient 调用 NameNode 的如下方法（需要加一些简单的检查）：

setReplication/rename/delete/exists（通过 getFileInfo 的返回值是否为空判断）/listPaths/getFileInfo/setPermission/setOwner/getDiskStatus/totalRawCapacity/totalRawUsed/datanodeReport/setSafeMode/refreshNodes/metaSave/finalizeUpgrade/mkdirs/getContentSummary/setQuota/setTimes。

在系统的 Web 展示中，通过 DFSClient 返回的数据，列表展示系统名、文件名、区块空间、权限和集群信息等。

3. 集群状态和节点控制

（1）查看集群运行状态

想要了解集群的运行状态，需要找到 FSNamesystem。FSNamesystem 是 NameNode 实际记录信息的地方，保存在 FSNamesystem 中的数据有：

- 文件名到数据块列表的映射（存放在 FSImage 和日志中）。

- 合法的数据块列表（上面关系的逆关系）。
- 数据块 DataNode 的映射（只保存在内存中，根据 DataNode 发过来的信息动态建立）。
- DataNode 上保存的数据块（上面关系的逆关系）。
- 最近发送过心跳信息的 DataNode（LRU）。

节点操作相关类图如图 12.42 所示。

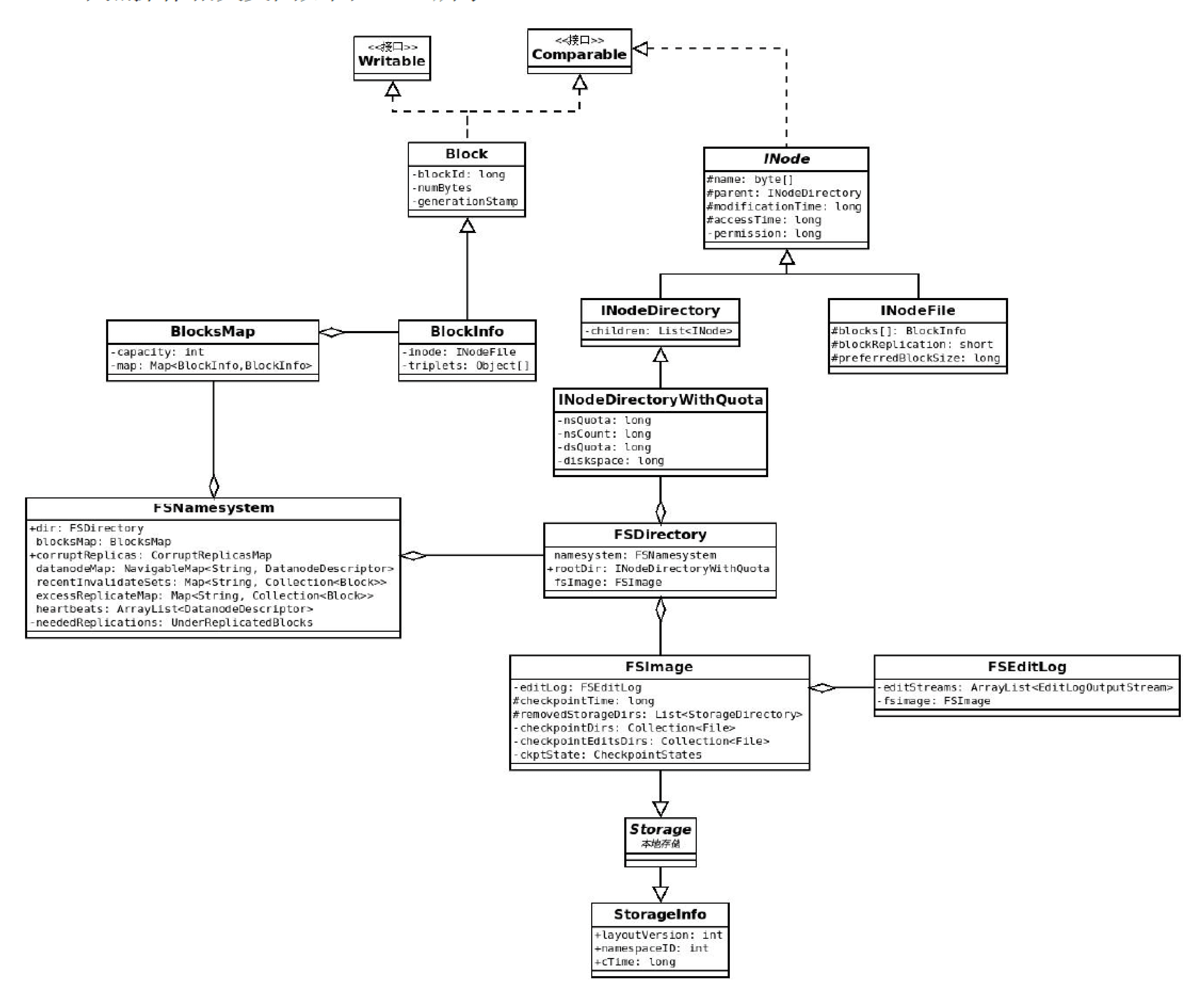


图 12.42　节点操作相关类图

- INode：它用来存放文件及目录的基本信息，即名称、父节点、修改时间、访问时间以及 UGI 信息等。
- INodeFile：继承自 INode，除 INode 信息外，还有组成这个文件的 Blocks 列表、重复因子和 Block 大小。
- INodeDirectory：继承自 INode，此外还有一个 INode 列表来组成文件或目录树结构。
- Block(BlockInfo)：组成文件的物理存储，有 BlockId、size 以及时间戳。
- BlocksMap：负责云文件系统中所有 Block 信息的管理，它维护了从 Block 到 INode 以及 DataNode 的映射。
- FSDirectory：保存文件树结构，HDFS 整个文件系统是通过 FSDirectory 来管理，也就是

维护整个云文件系统的层次目录信息，同时还负责元数据的加载以及持久化存储。

- FSImage：保存的是文件系统的目录树。
- FSEditlog：文件树上的操作日志。
- FSNamesystem：HDFS 文件系统管理。

（2）节点分类控制

FSNamenode 实现将文件名对应到相应的几台 datanode，首先是 filename→block 的实现。在运行时，这个对应关系一直放在内存中，也会放在硬盘中（fsimage 文件和相应的 edits.log 文件）。

在内存中，FSNamesystem 靠保存一棵目录树维持文件的目录结构，用到如下几个数据结构（如图 12.43 所示），这几个结构都和 Linux 的文件系统类似，INode 作为基类保存文件或者目录的名称，INodeDirectory 保存一个 INode 列表，表示目录下的文件或者目录，INodeFile 保存文件的信息，如 block 号、文件权限等。FSNamesystem 中的 dir 会保存一个 INodeDirectoryWithQuota 类型的 rootDir 保存根节点。

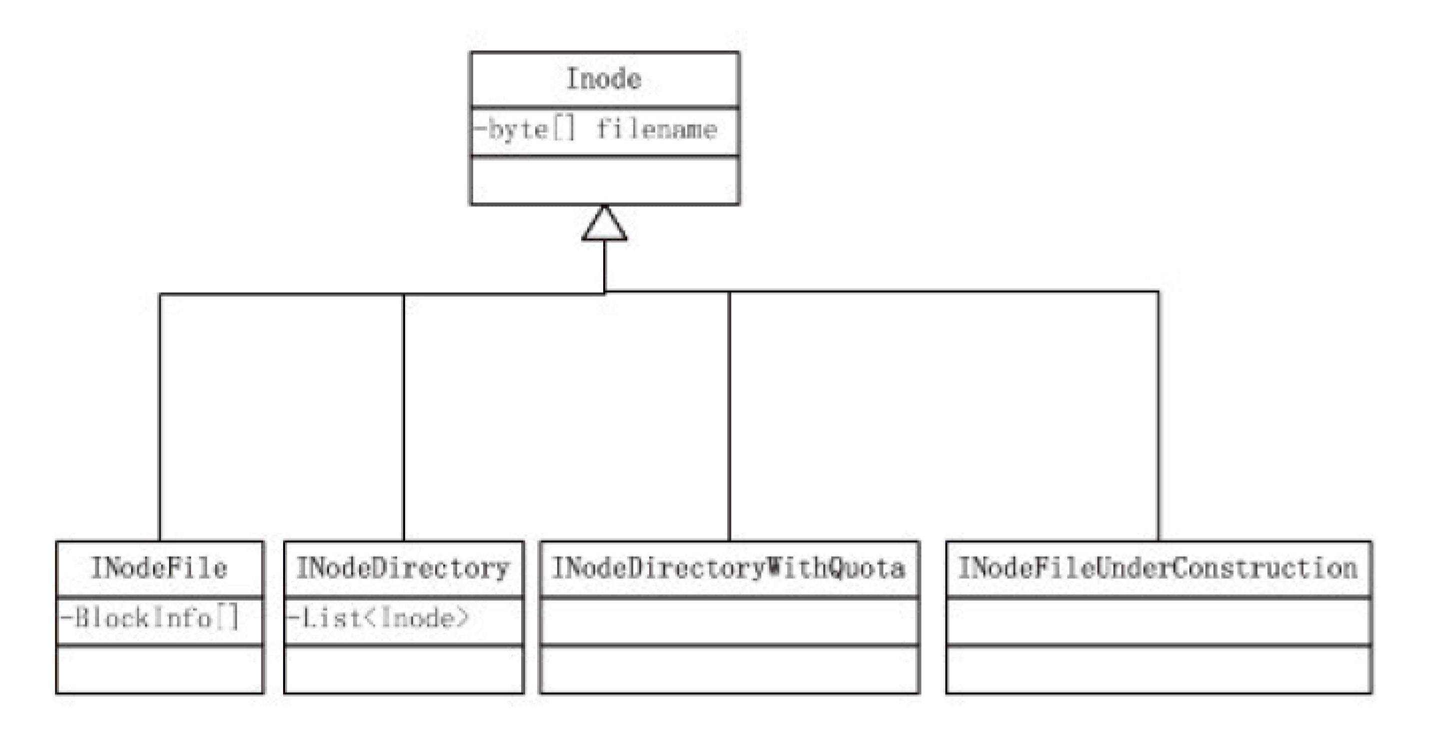


图 12.43　结点控制相关类图

在硬盘中，目录的信息以 fsimage 文件的形式存放，并将读入 fsimage 之后对文件结构的操作放在 edits.log 中。启动时，FSDirectory 调用 FSImage 的 LoadFSImage 方法和 LoadFSEdits 方法将 fsimage 和 edits.log 读入内存，合成为新的目录结构。Secondnamenode 会定期地读取 namenode 中的这两个文件，合成为一个新的 fsimage 文件，更新 namenode 中的 fsimage。FSImage 有升级/回滚/提交动作，FSImage 也能够管理多个 Storage，而且还能够区分 Storage 为 IMAGE（目录结构）/EDITS（日志）/IMAGE_AND_EDITS（前面两种的组合）。

filename→block 的关系中，通过 BlocksMap 类型的 blocksMap 元素保存在内存中，不在硬盘保存。它通过 datanode 的 blockReport 来收集构建。让 datanode 来告诉 namenode 哪一个 datanode 有什么 block。BlocksMap 中有一个 GSet<Block,BlockInfo>的元素，这涉及两个结构，Block 和 BlockInfo。Namenode 通过 FSNamesystem 靠前面结构中的 dir 和 blocksMap 这两个元素完成了两个重要功能。

FSNamesystem 统一表示内存元数据，在运行的云文件系统中，把节点分为三类，即活动节点、死亡节点和退役节点。通过 Web 选择，由 FSNameSystem 得到符合要求的节点列表，通过 JSPHelper 组织辅助，整合各节点信息数据、例如响应次数、分配区块个数，占用空间和剩余空间

及其百分比等，最后响应请求。

（3）元数据节点存储设计

类 Storage 保存了和存储相关的信息，它继承了 StorageInfo，应用于 DataNode 的 DataStorage，则继承了 Storage。Storage 可以包含多个根（参考配置项 dfs.data.dir 的说明），这些根通过 Storage 的内部类 StorageDirectory 来表示。在 HDFS 中，无论是 NameNode 节点还是 DataNode 节点都需要使用它们所在的本地文件系统来存储与自己相关的数据，如 NameNode 节点存储系统命名空间的元数据，DataNode 节点存储文件的数据块数据。总体类图如图 12.44 所示。

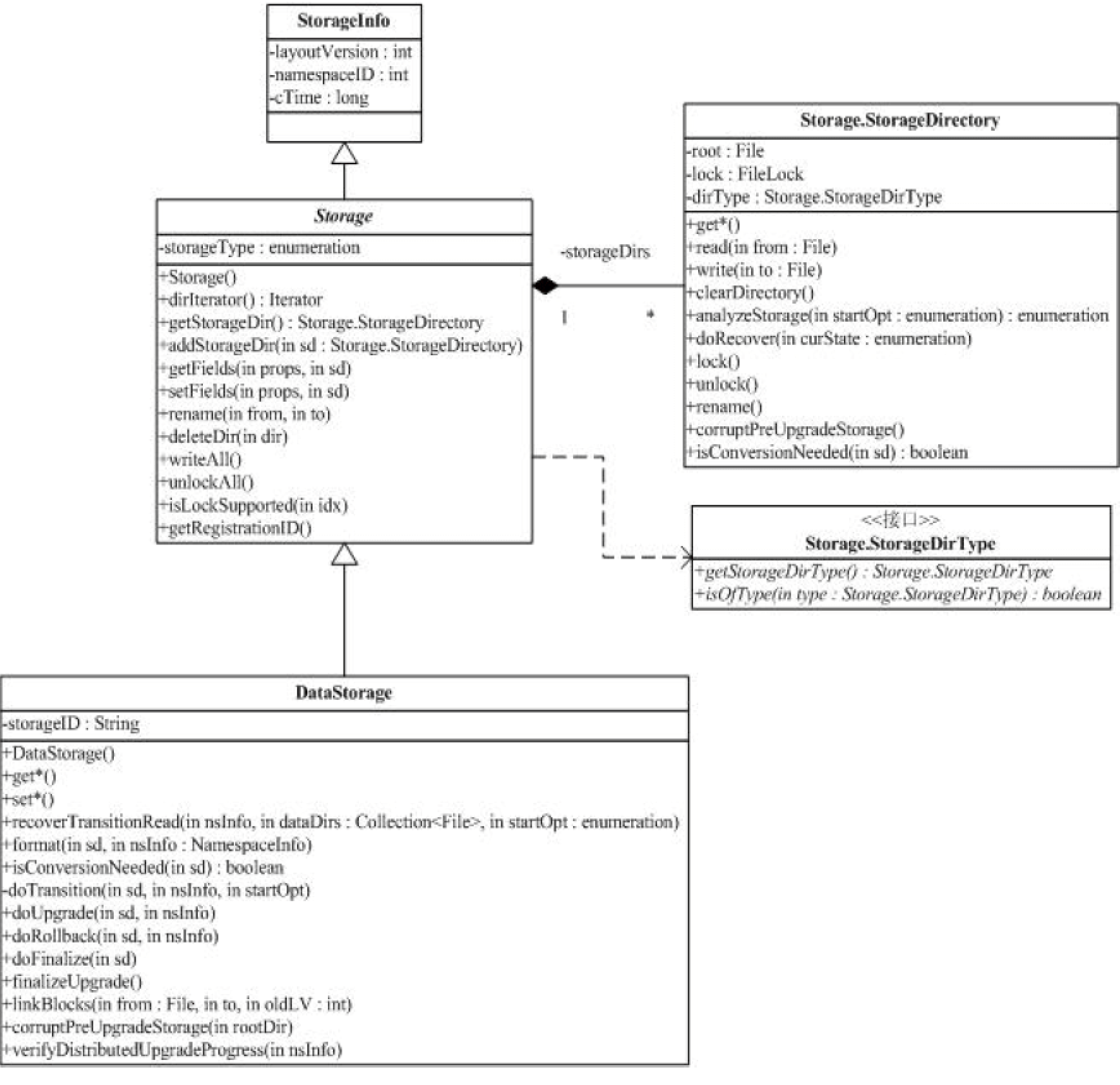


图 12.44　文件系统存储相关类图

对于 NameNode 节点或是 DataNode 节点，我们都可以为它们配置多个本地文件系统的存储路径，不同的是，NameNode 节点中的所有存储路径存储的数据基本上是一样的，而 DataNode 节点中的存储路径会分别存储不同的文件数据块。HDFS 对节点存储路径的实现被抽象成了一个 StorageDirectory 类。StorageDirectory 中最重要的方法是 analyzeStorage，它将根据系统启动时的参数和上面提到的一些判断条件，返回系统现在的状态。StorageDirectory 可能处于以下的某一个状态（与系统的工作状态一定的对应）：

- NON_EXISTENT：指定的目录不存在。

- NOT_FORMATTED：指定的目录存在但未被格式化。
- COMPLETE_UPGRADE：previous.tmp 存在，current 存在。
- RECOVER_UPGRADE：previous.tmp 存在，current 不存在。
- COMPLETE_FINALIZE：finalized.tmp 存在，current 存在。
- COMPLETE_ROLLBACK：removed.tmp 存在，current 存在，previous 不存在。
- RECOVER_ROLLBACK：removed.tmp 存在，current 不存在，previous 存在。
- COMPLETE_CHECKPOINT：lastcheckpoint.tmp 存在，current 存在。
- RECOVER_CHECKPOINT：lastcheckpoint.tmp 存在，current 不存在。
- NORMAL：普通工作模式。

StorageDirectory 处于某些状态是通过发生对应状态改变需要的工作文件夹和正常工作的文件夹来进行判断。Storage 类就很简单了，基本上都是对一系列 StorageDirectory 的操作，同时 Storage 提供一些辅助方法。DataStorage 是 Storage 的子类 专门应用于 DataNode。DataStorage 提供了 format 方法，用于创建 DataNode 上的 Storage，同时，利用 StorageDirectory、DataStorage 管理存储系统的状态。StorageDirectory 除提供保存节点数据的功能之外，还提供了对存储数据的粗粒度事务操作，如备份、恢复、提交等。

12.4.4　云文件系统主要功能截图

1. 管理员功能实现

（1）系统登录

系统管理员以账户名 admin、密码 admin 登录云文件系统的管理员页面，如图 12.45 和图 12.46 所示。



图 12.45　系统初始界面

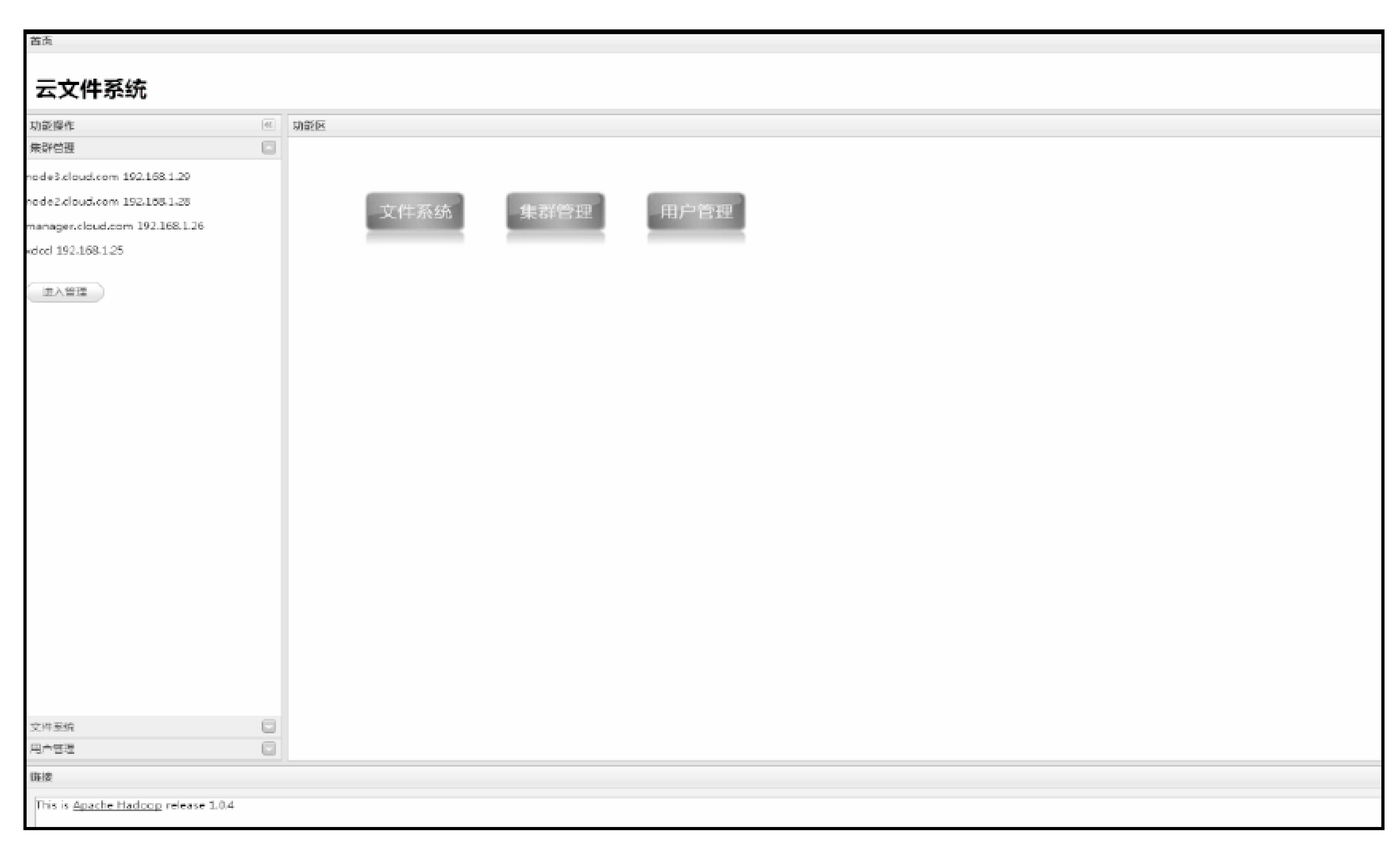


图 12.46　管理员页面

系统管理员有三个方面的功能，一是用户管理，二是集群管理，三是文件系统的管理。

（2）文件系统操作

管理员可以查看文件系统的所有文件，并拥有操作所有文件的权限。文件管理页面如图 12.47 所示。



图 12.47　文件管理页面

（3）集群管理

本系统中集群管理主要包括三个方面：集群扩展、删除节点以及修改集群的副本因子。

- *集群扩展*

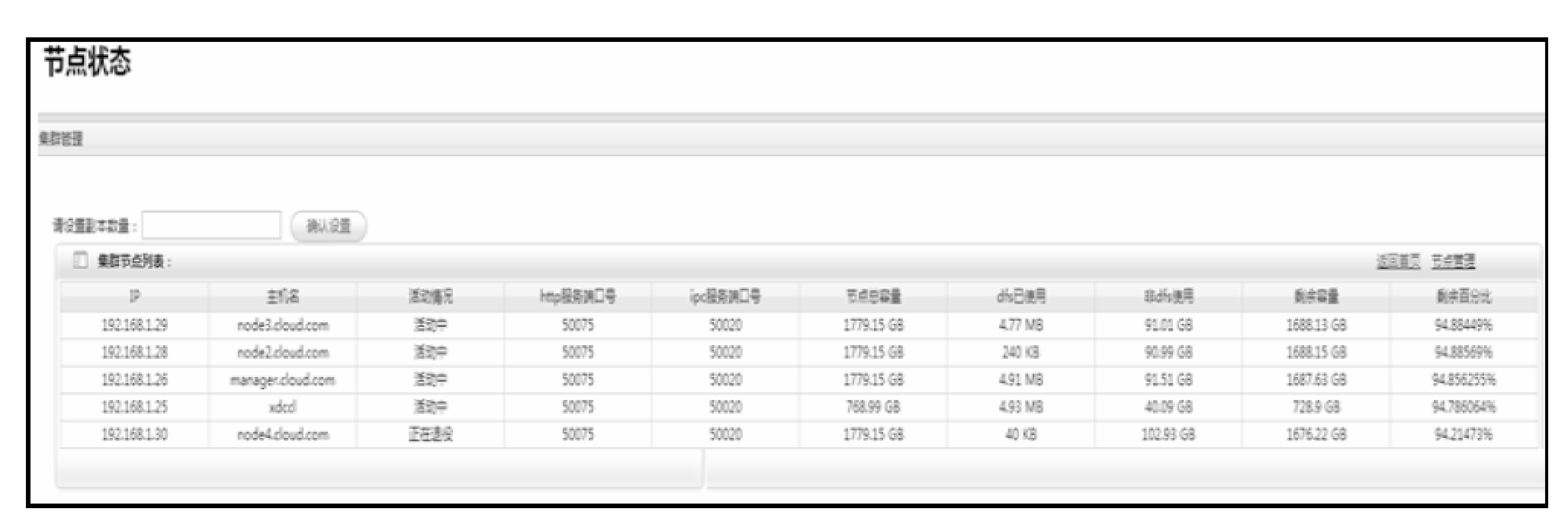
云文件系统支持向集群中动态增加机器的功能。在管理员页面单击集群管理按钮，进入如图 12.48 所示的集群管理页面。



IP	主机名	活动情况	http服务端口号	ipc服务端口号	节点总容量	dfs已使用	非dfs使用	剩余容量	剩余百分比
192.168.1.27	management.cloud.com	活动中	50075	50020	0	0	0	0	0.0%
192.168.1.25	xdccl	活动中	50075	50020	0	0	0	0	0.0%
192.168.1.28	node2.cloud.com	活动中	50075	50020	0	0	0	0	0.0%

图 12.48　集群管理页面

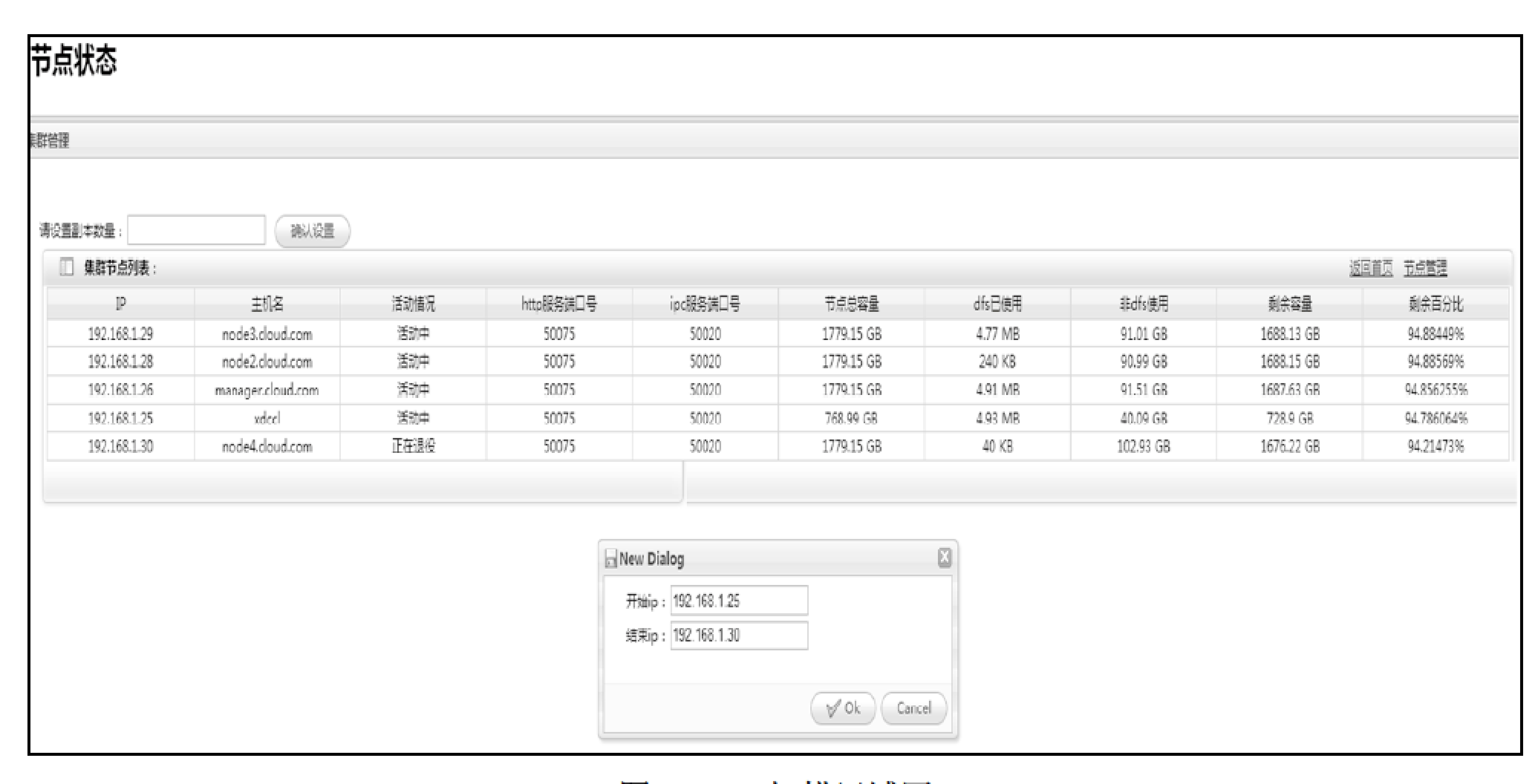
在集群管理页面中，详细列出了当前集群中每台数据节点的信息，包括主机 IP、主机名、活动情况以及存储信息等内容。在这个页面的右上角，单击“节点管理”进入集群扩展与减少机器的页面，如图 12.49 所示。



IP	主机名	活动情况	http服务端口号	ipc服务端口号	节点总容量	dfs已使用	非dfs使用	剩余容量	剩余百分比
192.168.1.29	node3.cloud.com	活动中	50075	50020	1779.15 GB	4.77 MB	91.01 GB	1688.13 GB	94.88449%
192.168.1.28	node2.cloud.com	活动中	50075	50020	1779.15 GB	240 KB	90.99 GB	1688.15 GB	94.88569%
192.168.1.26	manager.cloud.com	活动中	50075	50020	1779.15 GB	4.91 MB	91.51 GB	1687.63 GB	94.856255%
192.168.1.25	xdccl	活动中	50075	50020	768.99 GB	4.93 MB	40.09 GB	728.9 GB	94.786064%
192.168.1.30	node4.cloud.com	正在退役	50075	50020	1779.15 GB	40 KB	102.93 GB	1676.22 GB	94.21473%

图 12.49　集群扩展与删除节点页面

在这个页面中，单击扫描当前局域网的按钮，页面即可列出局域网中的每台机器，页面显示了集群中的节点和非集群中的节点，如图 12.50 所示。



IP	主机名	活动情况	http服务端口号	ipc服务端口号	节点总容量	dfs已使用	非dfs使用	剩余容量	剩余百分比
192.168.1.29	node3.cloud.com	活动中	50075	50020	1779.15 GB	4.77 MB	91.01 GB	1688.13 GB	94.88449%
192.168.1.28	node2.cloud.com	活动中	50075	50020	1779.15 GB	240 KB	90.99 GB	1688.15 GB	94.88569%
192.168.1.26	manager.cloud.com	活动中	50075	50020	1779.15 GB	4.91 MB	91.51 GB	1687.63 GB	94.856255%
192.168.1.25	xdccl	活动中	50075	50020	768.99 GB	4.93 MB	40.09 GB	728.9 GB	94.786064%
192.168.1.30	node4.cloud.com	正在退役	50075	50020	1779.15 GB	40 KB	102.93 GB	1676.22 GB	94.21473%

图 12.50　扫描局域网

用户可以在非集群的机器列表中勾选自己想要加入到集群中的机器，单击添加按钮。这时，页面会跳出让用户确认选择添加的对话框，单击确定即可。这样就完成了对集群的动态扩展。在扩展机器的过程中，同样要保证集群中的每台机器可以互相识别，即在 hosts 文件中加入每台机器的信息。

- 删除节点

在扫描完局域网中的机器后，页面同样会显示出集群中的每一台数据节点。用户可以在待删除机器的前面的小方框中打勾，然后单击删除按钮，这时页面中会跳出让用户确认删除节点的对话框，单击确定即可。因为在删除节点时，集群会将待删除节点上的数据转移到其他节点上，这个过程将根据待转移数据量的多少需要不等的时间。待数据转移完成之后，用户需要手动将删除节点上的数据删除。

- 修改副本参数

云文件系统的高可靠性部分依赖于每个文件都有若干副本存在，保证了在原数据因意外不可用时，副本还可以使用。因此这个副本数量也显得尤为重要，用户可以根据自己的需要设定不同的副本数量。在集群管理页面的左上角，有一个让用户设置副本数量的文本框，用户可以在这里填写相应的数值，单击设置按钮即可。云文件系统默认提供的副本数量是 3。

（4）用户管理

在管理员页面单击用户管理，即可进入用户管理的页面，如图 12.51 所示。在该页面，管理员可以修改已经存在的用户的信息、添加新用户以及删除当前存在的用户。



图 12.51　用户管理页面

2. 普通用户功能实现

用户创建后系统自动为用户生成 5 个文件夹（图片、应用、文档、视频和音乐），如图 12.52 所示。用户可以对自己的文件进行增、删、查，删除的文件将暂时保存在回收站，用户可以从回收站恢复近期删除的文件，文件在回收站中保存一天后自动清空。



图 12.52　用户文件系统界面

（1）上传文件

在用户当前目录单击上传文件，可以把本地文件上传到相对应的用户目录。如图 12.53 所示。



图 12.53　上传文件示意图

（2）新建文件夹

在用户当前目录创建新的文件夹，用户自由控制自己的文件目录和层次结构，如图 12.54 所示。



图 12.54　新建文件夹示意图

（3）文件检索

左侧的操作列表中，按类别对用户上传的文件进行划分，自动区分出图片、文档、音乐、视频和应用，用户名可以通过左侧操作直接得到自己所需文件而不用通过文件系统层次查找。通过右上方的搜索按钮和搜索框，可以按用户输入的内容进行模糊查询搜索，得到所有符合条件的结果并显示在页面列表中，如图 12.55 所示。

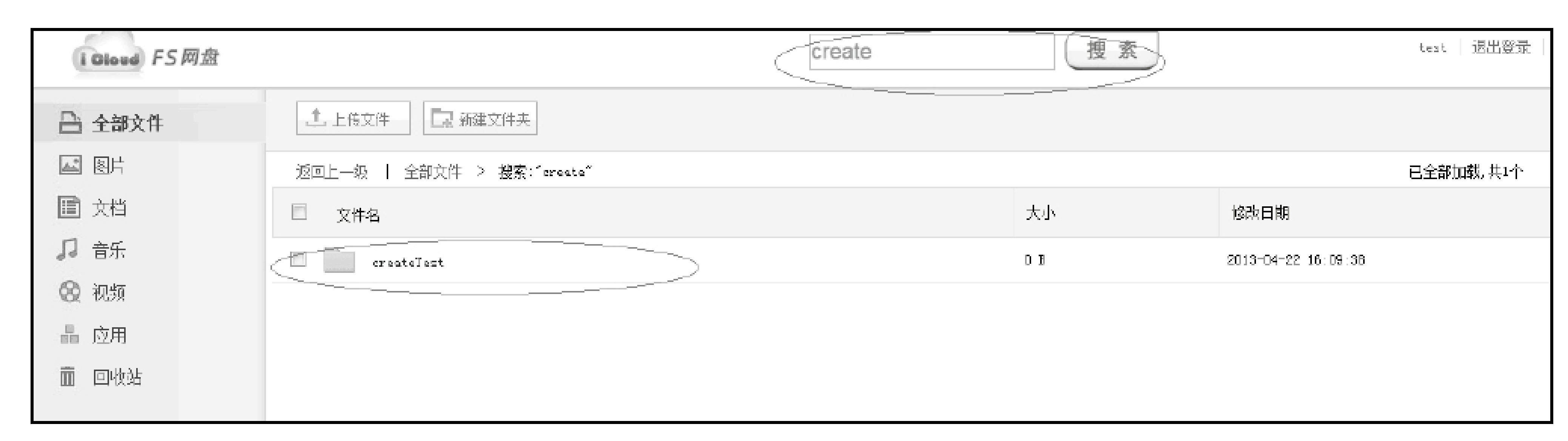


图 12.55　文件检索示意图

（4）文件删除和下载

文件操作中提供了删除和下载功能，单击删除可以进行文件删除（放入回收站），单击下载可以选择路径下载，如图 12.56 所示。

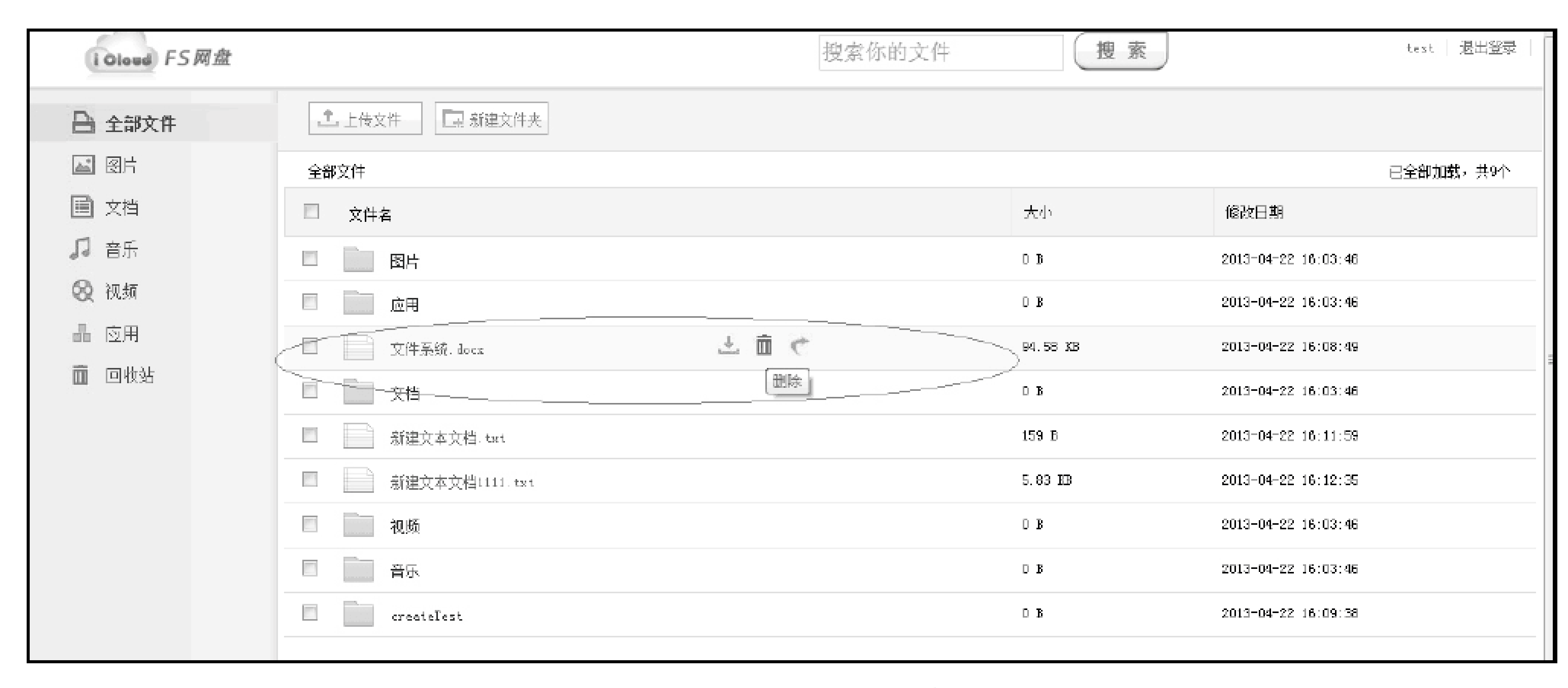


图 12.56　文件删除示意图

（5）文件恢复

回收站列表显示删除的文件或文件夹，相对应的操作有删除（彻底删除）和恢复功能，如图 12.57 所示。



图 12.57　文件恢复示意图

12.5 本章小结

本章着重讨论了基于 HDFS 的云文件系统的需求分析、总体设计、详细设计和实现过程，主要内容包括：

- 通过调研了解并完成了本系统的需求分析。
- 完成了本项目总体设计。
- 完成了系统各功能模块的设计和实现。
- 在各模块完成后，进行了功能测试工作。
- 测试后对本系统进行了优化。

本系统主要实现了网盘应用的基本功能，给出了一种基于云文件系统的数据存储、操作和检索的实现思路，希望对读者有所帮助。